RL-TR-97-173
Interim Report
October 1997

RT_STAP: REAL-TIME SPACE-TIME ADAPTIVE PROCESSING BENCHMARK

The MITRE Corporation

Kenneth C. Cain, Jose A. Torres and Ronald T. Williams

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-173 has been reviewed and is approved for publication.

APPROVED:

RALPH KOHLER
Project Engineer

FOR THE DIRECTOR:

JAMES W. CUSACK, Acting Director
Surveillance & Photonics Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/OCSS, 26 Electronic Pky, Rome, NY 13441-4514. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Oct 97	Interim Sep 96 - Feb 97

4. TITLE AND SUBTITLE

RT_STAP: REAL-TIME SPACE-TIME ADAPTIVE PROCESSING BENCHMARK

5. FUNDING NUMBERS

C - F19628-94-C-0001
PE - 63789F
PR - MOIE
TA - 74
WU - 11

6. AUTHOR(S)

Kenneth C. Cain, Jose A. Torres and Ronald T. Williams

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The MITRE Corporation
202 Burlington Rd.
Bedford, MA 01730-1420

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory/OCSS
26 Electronic Pky
Rome, NY 13441-4514

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-97-173

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Ralph Kohler, OCSS, 315-330-2016

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

N/A

13. ABSTRACT *(Maximum 200 words)*

This report provides a specification of the RT _STAP benchmark for evaluating the application of scalable high performance computers to the real-time implementation of space-time adaptive processing (STAP) techniques on embedded platforms. STAP is an adaptive processing technique used to support clutter and interference cancellation in airborne radars. The RT_STAP benchmark is an example of a compact application benchmark that uses a real-time design-to-specification methodology. The scalability study outlined in the RT_STAP benchmark varies the sophistication and computational complexity of the adaptive algorithms to be implemented. The benchmark provides hard, medium, and easy benchmark cases based upon three post-Doppler adaptive processing algorithms: higher-order Doppler-factored STAP, first-order Doppler-factored STAP, and post-Doppler adaptive Displaced Phase Center Antenna (CDPCA).

14. SUBJECT TERMS	15. NUMBER OF PAGES
Space-Time Adaptive Processing (STAP), Benchmark, Post-Doppler Adaptive Processing Algorithms, Higher-Order Doppler-Factored STAP	74
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Abstract

This report provides a specification of the RT_STAP benchmark for evaluating the application of scalable high performance computers to the real-time implementation of space-time adaptive processing (STAP) techniques on embedded platforms. STAP is an adaptive processing technique used to support clutter and interference cancellation in airborne radars. The RT_STAP benchmark is an example of a compact application benchmark that uses a real-time design-to-specification methodology. The scalability study outlined in the RT_STAP benchmark varies the sophistication and computational complexity of the adaptive algorithms to be implemented. The benchmark provides hard, medium, and easy benchmark cases based upon three post-Doppler adaptive processing algorithms: higher-order Doppler-factored STAP, first-order Doppler-factored STAP, and post-Doppler adaptive Displaced Phase Center Antenna (DPCA).

Acknowledgments

The authors would like to thank Dr. Richard A. Games for his contributions to our work. Dr. Games provided numerous comments and suggestions that had an invaluable impact on the content and format of this report. We would also like to thank Geoffrey Hendrey, Perry Lee and Ana Torres for their contributions to the development of the C code that provides a sequential implementation of this specification.

This work was supported by the United States Air Force Electronic Systems Center and performed under MITRE Mission Oriented Investigation and Experimentation (MOIE) Project 03967426 of contract F19628-94-C-0001, managed by Rome Laboratory/OCSS.

Table of Contents

List of Figures

List of Tables

Section 1

Introduction

A benchmarking methodology for assessing the use of scalable high performance computing (HPC) in real-time embedded applications has been proposed (Games, 1996). This methodology assesses both the ability of an HPC architecture to implement existing processing requirements under strict size, weight, and power constraints, along with the scalability of the architecture to meet future processing needs. It adopts a "design-to-specification" approach that utilizes both functional and timing specifications. Using this benchmarking approach, competing architectures can be evaluated by determining the smallest sized scalable architecture that meets the required timing and functional specifications. The minimum sized solutions can be compared using common measures of cost, size, weight, and power. In practice, benchmarks are applied at various levels including low-level benchmarks that focus on key elements of the HPC system, intermediate-level benchmarks involving computation and communication kernels, and high-level compact applications consisting of a few thousand lines of code. This report applies the methodology to a series of compact applications associated with the use of advanced signal processing techniques in airborne moving target indication (MTI) radar systems.

In particular, this report provides a specification of the `RT_STAP` benchmark. This benchmark is concerned with the implementation of space-time adaptive processing (STAP) techniques used to support clutter and interference cancellation in airborne radars. STAP algorithms combine both spatial and temporal adaptive processing to cancel Doppler-spread clutter and interference contained in radar signals measured by an airborne antenna array. These algorithms have the potential to significantly improve the functional performance of the radar system while requiring little or no modification to the basic radar design (e.g., increase in array size or modification of the signal waveform). These techniques are of particular interest given that future upgrades to a number of existing surveillance platforms may incorporate some form of STAP.

The computational complexity of more advanced STAP algorithms will easily overwhelm the computational capabilities of processors used on current airborne platforms. Consequently, as platforms incorporate STAP algorithms into their radar signal processing, processor upgrades will be required. In addition to supporting the computational needs of current processing techniques, these upgrades must scale to the requirements of future processing methods in a manner that meets the strict size, weight, and power limitations associated with an airborne platform. Our goal is to provide a benchmarking methodology that could be used to evaluate HPC architectures intended to support future STAP upgrades.

The potential for STAP algorithm upgrades is reflected in the scalability study of the RT_STAP benchmark. Typically the scalability of a parallel architecture has been evaluated by varying the size of the data processing problem (i.e., varying the amount of data to be processed) for a given algorithm and architecture. However, in developing the RT_STAP benchmark, an alternative approach was considered. In this case, scalability is evaluated by varying the algorithm complexity for a given processing problem. Separate benchmark cases for signal processing techniques of varying sophistication are developed. These benchmark cases provide varying levels of functional performance at the cost of increased computational throughput requirements. The proposed RT_STAP benchmark evaluates the ability of a parallel processing architecture to scale as a function of algorithm complexity.

The scalability study for the RT_STAP benchmark involves three levels of complexity: easy, medium, and hard. The easy benchmark case corresponds to the post-Doppler adaptive Displaced Phase Center Antenna (DPCA) algorithm and requires a computational throughput of 0.60 billion floating-point operations[1] per second (Gflop/s). This case represents technology used in current radar systems for performing a similar function to STAP. It can be implemented on a variety of high performance computers with little difficulty. The medium benchmark case corresponds to first-order Doppler-factored STAP and requires 6.46 Gflop/s. The hard benchmark case corresponds to an implementation of third-order Doppler-factored STAP and requires a throughput of 39.81 Gflop/s.

---

[1] A floating-point operation is defined to be the addition, subtraction, or multiplication of two real numbers in floating-point representation.

The RT_STAP benchmark includes the implementation of the data preprocessing typically performed before the application of STAP algorithms. Preprocessing typically includes: conversion of the received radar signals from video to in-phase and quadrature (I/Q) samples at baseband, array calibration, and pulse compression. In the past, preprocessing has been implemented using special-purpose hardware. However, we view this processing as a potential application of HPC technology, particularly when specialized processing nodes are allowed. Implementing the preprocessing functions within the HPC communication fabric should significantly reduce the number of interfaces, thus simplifying the system.

In this report, we use the Multi-Channel Airborne Radar Measurement (MCARM) system as an example for determining the throughput requirements of both the preprocessing and the STAP algorithms. The MCARM data collection system was developed for Rome Laboratory by the Westinghouse Electronic Systems Group (now Northrup Grumman, Electronic Sensors and Systems Division, Baltimore, Maryland) for the purpose of collecting and recording L-band radar returns transmitted from an airborne platform. The data measured by the MCARM data collection system can be used to evaluate the ability of STAP techniques to cancel Doppler-spread clutter and interference. Rome Laboratory has collected a number of useful data sets and has made them available for processing. We use a selected data set to provide the input for sequential software that implements RT_STAP.

The algorithms chosen for the benchmark cases in RT_STAP were selected to provide a means for evaluating the performance of current and future HPC architectures and to support the development, demonstration, and evaluation of our benchmarking methodology. We do not endorse the use of a particular STAP algorithm in any particular system. Furthermore, this particular set of benchmarks is not meant to suggest a particular migration path for any airborne radar platform. While algorithm performance was and will continue to be a consideration in selecting the algorithms to be benchmarked, we make no judgments concerning the relative functional performance of these techniques beyond those drawn from the existing literature and referenced in the text.

A real-time benchmark specification (Games, 1996) consists of a functional specification, a timing specification, a scalability study, sequential code that implements the functional specification, and implementation guidelines. Section 2 describes the functional specification of the preprocessing common to all adaptive processing techniques considered. Section 3 describes the functional specification of the post-Doppler adaptive algorithms.

Section 4 describes the real-time period and latency requirements of the RT_STAP benchmark along with the scalability study. The period determines the computational throughput required and follows easily from the sensor front end, in this case from the duration of the coherent processing interval (CPI) of the MCARM system. The strictness of the latency requirement determines the difficulty of the parallel implementation. This requirement is often system specific, and so to cover the two extremes we specify two latency cases: short (in terms of a number of CPIs) and unlimited. The scalability study described in Section 4 calculates the throughput requirement of the easy, medium, and hard cases using the MCARM system parameters.

Section 5 describes the sequential implementation of the preprocessing and STAP algorithms that is provided with the RT_STAP benchmark. Section 6 gives the implementation guidelines and reporting requirements for the RT_STAP benchmark. Section 7 summarizes and concludes the report.

Before concluding this section, a few words on notation are in order. Throughout the document we use lower case letters to denote scalar variables and indices (e.g., $x(i)$ and i). Upper case letters are used to denote the upper and/or lower bounds of an index (e.g., $n = 0,\ldots,N$). Column vectors are denoted by lower case letters overscored by a right-pointing arrow (e.g., $\vec{x}$), while matrices are represented by upper case letters also overscored by a right-pointing arrow (e.g., $\vec{X}$). To denote that the variable x is a function of the index i we use the notation $x(i)$. This same notation is used to index vectors and matrices.

A number of operators are used throughout the report. The superscript "T" denotes the transpose operator while "H" is used to denote Hermitian transpose. The superscript "*" denotes the complex conjugate operator. We make use of two functions denoted by $\lfloor \cdot \rfloor$ and

$\lceil \cdot \rceil$. The first function corresponds to the largest integer smaller than or equal to the contents of the operator, while the second function corresponds to the smallest integer larger than or equal to the contents of the operator.

Section 2

Preprocessing Functional Specification

Before applying space-time adaptive processing algorithms to data samples measured by an airborne antenna array, a significant amount of preprocessing must be performed. The type of preprocessing varies between systems; however, there are generally three major functions: (1) video-to-I/Q conversion, (2) array calibration, and (3) pulse compression. Figure 2-1 shows a block diagram of the STAP preprocessing. These functions are applied to the A/D data samples independently across the L channels. Complex data samples are passed between each component of the preprocessing with the data at the output of the pulse compression functional block forming the input to the STAP.

To support the STAP algorithm, a data cube (Figure 2-2) corresponding to the L channels, P pulse repetition intervals (PRIs), and N time samples per PRI, must be processed. This data cube corresponds to a single CPI of the radar system. On input, these data samples will be real-valued integers. We define $x(l,p,n)$ to be a real data sample corresponding to the n-th time sample from the p-th PRI of the l-th channel of the CPI. The following subsections describe the functionality of the preprocessing that is to be performed on this data cube. Each subsection specifies the component functionality to be implemented and provides estimates of the computational complexity for processing a single CPI. For all operations described in the subsequent sections, the data samples from each channel and each PRI are processed independently.

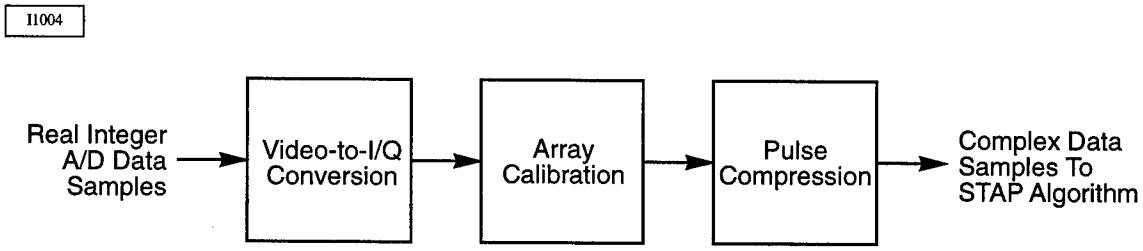

Figure 2-1. Preprocessing Block Diagram for a Single-Array Channel

I1005

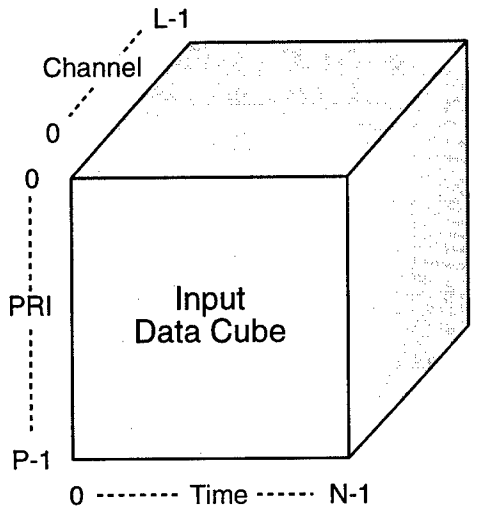

Figure 2-2. Input Data Cube for a Single CPI

2.1 Video-to-I/Q Conversion

For many array processing systems, digital receivers are used to demodulate the signal to baseband and generate digital samples using a sampling rate at or slightly above the Nyquist rate. In some cases, digital data is generated at an intermediate frequency (IF) and sampled at a higher rate than required to accurately represent the baseband signal. The Rome Laboratory MCARM system, for example, provides IF data to the preprocessor. For these systems, the digital data must be demodulated to baseband, lowpass filtered, and decimated to a lower sample rate. These processes are implemented in the video-to-I/Q conversion functional block.

Demodulation to baseband is achieved by multiplying the data by demodulation coefficients (i.e., complex sinusoid) that translate the signal to baseband. The frequency translated samples are then processed by a lowpass filter to remove aliased frequency components. Sample rate conversion decimates the data by simply choosing the samples corresponding to the desired sample rate. For complex data, this sample rate must be at least

as large as the bandwidth of the signal. Typically, the data is slightly over-sampled to allow for array calibration; but in general, over-sampling is minimized to limit the data rate. The complex data generated at the output of the video-to-I/Q functional block forms the input to the calibration process.

If we let f_{IF} denote the center frequency of real data samples, f_{A-D} be the sampling frequency of the data, and $h_d(p,n)$ be the complex demodulation coefficients, then the output after frequency translation is:

$$x_0(l,p,n) = h_d(p,n)x(l,p,n), \tag{2-1}$$

where $n = 0,\ldots,N-1$, $p = 0,\ldots,P-1$, $l = 0,\ldots,L-1$, and $h_d(p,n)$ is given by:

$$h_d(p,n) = 2\exp\{-j2\pi f_{IF}(n+pN)/f_{A-D}\}. \tag{2-2}$$

Implementation of the demodulation to baseband function requires $2 \cdot N$ floating-point operations per channel and PRI, resulting in a total of $L \cdot P \cdot (2 \cdot N)$ floating-point operations.

Anti-aliasing is then accomplished with a finite impulse response (FIR) lowpass filter. The length of the filter is defined to be K_a and the real-valued filter coefficients are denoted by $h_a(k)$ for $k = 0,\ldots,K_a-1$. The output of the filter corresponds to the discrete linear convolution of the real filter coefficients with the complex data:

$$x_1(l,p,n) = \sum_{k_a=0}^{K_a-1} h_a(k_a)x_0(l,p,n-k_a), \tag{2-3}$$

for $n = 0,\ldots,N-1$, $p = 0,\ldots,P-1$, and $l = 0,\ldots,L-1$. When implementing this filter we assume that $x_0(l,p,n) = 0$ for $n < 0$. In general, we will design the FIR filter to have linear phase. In this case, the filter coefficients must satisfy the condition:

$$h_a(n) = h_a(K_a-1-n). \tag{2-4}$$

We can take advantage of this property to reduce the computational complexity of implementing equation (2-3). Note that the FIR filter coefficients are normalized such that the inner product $\vec{h}_a^H \vec{h}_a$ equals unity.

After applying the lowpass filter, the sample rate conversion function decimates the data to achieve the desired data rate. We assume the data is to be decimated by an integer value, D, defined to be the ratio of the sampling rate of the data on input to the preprocessor and the desired sampling rate after conversion. The final output of the conversion process is:

$$x_2(l,p,n_D) = x_1(l,p,n_D \cdot D), \tag{2-5}$$

where $n_D = 0,\ldots,N_D - 1$, $p = 0,\ldots,P-1$, $l = 0,\ldots,L-1$, and $N_D = \lfloor N/D \rfloor$.

In an attempt to minimize computational complexity, we considered two methods for implementing the lowpass filter: discrete linear convolution and fast convolution based on the fast Fourier transform (FFT). Due to the decimation that takes place as part of this process, we found that discrete linear convolution provides the most efficient implementation for this application. Note from equation (2-5) that only one of every D output samples of equation (2-3) needs to be explicitly computed. As a result, it is most efficient to compute equation (2-3) explicitly for the subset of data samples that form the output of the data rate conversion process. By taking advantage of the lowpass filter's linear phase property described in (2-4), we can reduce the number of floating-point operations required to implement the filter. Using this result, the FIR filtering and decimation function requires $3 \cdot K_a \cdot N_D$ floating-point operations per channel and PRI, resulting in a total of $L \cdot P \cdot (3 \cdot K_a \cdot N_D)$ floating-point operations. Note that the number of operations reflects real-valued filter coefficients.

2.2 Array Calibration

Array calibration is essential to performing the spatial-temporal filtering associated with STAP. For example, to maintain high antenna gain in the direction of the desired signal, constraints are applied to the adaptive weights. The constraints help prevent the adaptive

processing from nulling signals located in the mainbeam of the antenna. To define these constraints the antenna response must be known. In general, however, variations in the response of the antenna elements, receivers, and other components of the array over time introduce unknown amplitude and phase variations in the data. For wideband signals, these variations are often a function of frequency and change across the passband of the receivers. These frequency-dependent variations can also affect the ability of a STAP algorithm to adequately null undesired interference. As a result, successful application of most STAP algorithms requires that the antenna response be measured and equalized across all channels.

Calibration can be achieved by applying an FIR filter to the data with filter coefficients designed to equalize the antenna response. Filter coefficients are typically determined off-line from measurements of the antenna response to known signals. The process associated with defining equalization filter coefficients is not typically part of the preprocessing. The coefficients associated with these filters vary across antenna elements; however, we assume the filter length, K_c, is constant across the array. We assume that the filter coefficients are precomputed and do not need to be determined as part of the preprocessing. If we denote the filter coefficients by $h_c(l,k)$, then the output of the array calibration process can be written as:

$$x_3(l,p,n_D) = \sum_{k_c=0}^{K_c-1} h_c(l,k_c)\, x_2(l,p,n_D - k_c), \tag{2-6}$$

where $n_D = 0,\ldots,N_D-1$, $p = 0,\ldots,P-1$, and $l = 0,\ldots,L-1$. We assume that $x_2(l,p,n) = 0$ for $n < 0$. The most efficient implementation for (2-6) is to use either overlap-save or overlap-add fast convolution techniques. Additional savings can be obtained by combining the implementation of the calibration process with pulse compression. We discuss the combined implementation of these two processes in Section 2.4.

2.3 Pulse Compression

By employing pulse compression techniques, radar systems can transmit relatively long pulses with low peak power to achieve high signal energy and improved detection

performance. By also employing a properly designed phase or frequency modulated signal, these systems can obtain resolution comparable to that achieved by systems that employ shorter pulses (Chapter 15, Eaves and Reedy, 1987, pp. 420-424; Skolnik, 1980). Pulse compression is implemented by passing the received data through an FIR filter with coefficients matched to the signal waveform. After being passed through this "matched filter," the pulse will have a duration equivalent to the inverse of the transmitted signal bandwidth. To reduce range sidelobes and improve range resolution, a taper is often applied to the coefficients of the matched filter. The complex data samples generated at the output of the pulse compression filter form the input to the STAP. If we let $h_{\mathrm{p}}(k),\ k=0,\ldots,K_{\mathrm{p}}-1$, denote the K_{p} FIR filter coefficients used in pulse compression, then the output samples after pulse compression can be written as:

$$x_4(l,p,n_{\mathrm{D}}) = \sum_{k_{\mathrm{p}}=0}^{K_{\mathrm{p}}-1} h_{\mathrm{p}}(k_{\mathrm{p}})\, x_3(l,p,n_{\mathrm{D}}-k_{\mathrm{p}}), \tag{2-7}$$

where $n_{\mathrm{D}}=0,\ldots,N_{\mathrm{D}}-1,\ p=0,\ldots,P-1,\ l=0,\ldots,L-1$, and $x_3(l,p,n)=0$ for $n<0$. After pulse compression is complete, the variable n_{D} corresponds to the range index.

In general, the filter coefficients are matched to the transmitted signal with a taper applied to reduced range sidelobes. The coefficients can be written in the form:

$$h_{\mathrm{p}}(k) = s_{\mathrm{p}}(k)\, w(k), \tag{2-8}$$

where $s_{\mathrm{p}}(k)$ represents the k-th matched signal sample and $w(k)$ corresponds to the k-th weight of the taper. The choice of signal waveform and taper is heavily dependent upon the application. Note that the filter coefficients are normalized such that the inner product $\vec{h}_{\mathrm{p}}^{\,\mathrm{H}}\,\vec{h}_{\mathrm{p}}$ equals unity. In Appendix A we give an example of typical real-valued filter coefficients for Rome Laboratory's MCARM platform. As with the calibration process, the most effective implementation of pulse compression uses one of two well-known fast convolution techniques. As we noted in the previous section, it is most efficient to combine implementation of the calibration and pulse compression processes. The combined implementation of these functions is described in the next section.

2.4 Implementation of Calibration and Pulse Compression

The combined implementation of calibration and pulse compression requires computation of the following expression:

$$x_4(l,p,n_D) = x_2(l,p,n_D) * h_c(l,n_D) * h_p(n_D) = x_2(l,p,n_D) * h_{cp}(l,n_D), \qquad (2\text{-}9)$$

where "*" denotes discrete linear convolution over the index n_D. We assume that the convolution of the filter coefficients is performed off-line to produce a combined filter response of $h_{cp}(l,k)$, corresponding to a sequence of length $K_{cp} = K_c + K_p - 1$. The combined coefficients can be computed using:

$$h_{cp}(l,n_D) = \sum_{k_c=0}^{K_c-1} h_c(l,k_c)\, h_p(n_D - k_c), \qquad (2\text{-}10)$$

for $n_D = 0,\ldots,K_{cp}-1$ and $l = 0,\ldots,L-1$. Note that $h_c(l,k) = 0$ for $k < 0$ or $k \geq K_c$ and $h_p(k) = 0$ for $k < 0$ or $k \geq K_p$. The output of the preprocessing corresponds to the discrete linear convolution of the combined filter coefficients and the set of length N_D sequences of data samples, $x_2(l,p,n_D)$, for $p = 0,\ldots,P-1$ and $l = 0,\ldots,L-1$.

Fast convolution techniques based upon either the overlap-add or overlap-save methods represent the most efficient implementation of the linear convolution $x_2(l,p,n_D) * h_{cp}(l,n_D)$ over the index n_D. Since the choice of fast convolution algorithm might be system dependent, we leave it for the developer to select the most appropriate method. Detailed descriptions of these techniques are available in the literature (Oppenheim and Schafer, 1975, pp. 110-115). In order to discuss design tradeoffs associated with the implementation and to estimate computational complexity, the subsequent discussion provides an implementation of the combined filtering process using the overlap-save method.

The overlap-save method implements a discrete circular convolution of segments of the complex data and the filter coefficients, followed by a selection of the part of the circular convolution corresponding to the linear convolution of the two sequences. To begin, we

augment the complex data $x_2(l,p,n_D)$ with $K_{cp}-1$ leading zeros. The augmented data sequence is then divided into B overlapping segments of length $\tilde{R}+K_{cp}-1$, where $\tilde{R}$ and B are related through the expression $B=\lceil N_D/\tilde{R}\rceil$ and $\tilde{R}$ is the length of the discrete linear convolution. Each segment is overlapped by $K_{cp}-1$ samples. Each data block is circularly convolved with the sequence of filter coefficients, $h_{cp}(l,n_D)$, using FFT techniques. FFTs are applied to both the data block and the sequence of filter coefficients with both sequences zero padded so that the length of the FFT is a power of two. Transformation of the filter response can be performed off-line.

$\tilde{R}$ is selected to minimize computation count. To minimize zero padding and improve the efficiency of the processing, $\tilde{R}$ is selected so that $\tilde{R}+K_{cp}-1$ is a power of two. The length of the FFT, R, is then set to $\tilde{R}+K_{cp}-1$. As a result of this choice of $\tilde{R}$, only the last data block will need to be zero padded. Zero padding is avoided completely when $N_D/\tilde{R}$ is an integer. It is also desirable to make $\tilde{R}$ as large as possible to improve the efficiency of the implementation. These two design goals are not always compatible and tradeoffs must be made depending upon the number of samples per PRI.

Once the FFTs are computed, the transformed sequences are multiplied and an inverse FFT is applied to the result to obtain the time-domain representation of the circular convolution. The first $K_{cp}-1$ samples are discarded and the remaining samples from the B data segments are assembled to form the final output of the preprocessing.

Implementation of the FFTs and inverse FFTs used to compute the convolution requires $5\cdot R\cdot\log_2 R$ floating-point operations per FFT or inverse FFT (Oppenheim and Schafer, 1975, p. 305). Multiplication of the sequences in the frequency domain requires $6\cdot R$ floating-point operations per data block. The total operation count for calibration and pulse compression processing of the entire data cube is $L\cdot P\cdot B\cdot(10\cdot R\cdot\log_2 R+6\cdot R)$, where $R=K_c+K_p+\tilde{R}-2$ and $B=\lceil N_D/\tilde{R}\rceil$. The number of data blocks, B, and the length of the FFT, R, are selected to minimize the computational complexity of the implementation, and their values vary as a function of the number of time samples per PRI.

2.5 Summary

In this section, functional specifications were presented for the three components of the preprocessing: video-to-I/Q conversion, array calibration, and pulse compression. We also evaluated the computational requirements of the preprocessing. During this analysis, we observed that computational complexity can be minimized by combining the implementation of array calibration and pulse compression using fast convolution techniques. Table 2-1 below summarizes the results of our analysis.

Table 2-1. Preprocessing Computation Counts

Function	Operation Count
Video-to-I/Q Conversion	$L \cdot P \cdot (2 \cdot N + 3 \cdot K_a \cdot N_D)$
Calibration and Pulse Compression	$L \cdot P \cdot B \cdot (10 \cdot R \cdot \log_2 R + 6 \cdot R)$; $R = K_c + K_p + \tilde{R} - 2;\ \ B = \lceil N_D / \tilde{R} \rceil$

Below we have defined the parameters used in the expressions for the preprocessing computation counts:

L: Number of channels

P: Number of PRIs per CPI

N: Number of samples per PRI before decimation

D: Decimation factor

N_D: Number of samples per PRI after decimation; $N_D = \lfloor N / D \rfloor$

K_a: FIR filter length used for anti-aliasing in video-to-I/Q conversion

K_c: FIR filter length used in array calibration

K_p: FIR filter length used in pulse compression

R: FFT size (power of 2) used by the overlap-save fast convolution method

$\tilde{R}$: Discrete linear convolution length in array calibration and pulse compression

B: Number of blocks in the overlap-save fast convolution method; $B = \lceil N_D / \tilde{R} \rceil$.

Section 3

Post-Doppler Adaptive Processing Functional Specifications

Modern airborne radar systems must have the capability to detect targets having relatively small cross-sections in the presence of strong clutter and interference. Due to the motion of the radar platform, the clutter returns will have a non-zero Doppler shift that is dependent upon the location of the clutter source with respect to the radar's receiving antenna array. Consequently, clutter arriving in the antenna sidelobes may have an extensive Doppler spread that encompasses the Doppler frequency of a target signal to be detected. The sidelobe clutter cannot be removed using conventional processing techniques that apply temporal filtering to suppress clutter returns arriving around zero Doppler frequency. While conventional beamformers can provide spatial filtering to limit the impact of interference and clutter arriving in the sidelobes of the antenna, it becomes difficult and expensive to develop fixed beamformers having sidelobe levels low enough to suppress strong sidelobe clutter. Space-time adaptive processing algorithms have been developed to directly address cancellation of Doppler-spread clutter (Brennan and Reed, 1973).

Space-time adaptive processing techniques take advantage of both the spatial and Doppler diversity of target signal returns, clutter, and interference to extract the desired signal. These techniques adaptively combine samples from multiple channels and pulses (i.e., PRIs) to null clutter returns and interference. Processing data from multiple channels provides the radar an opportunity to control the spatial response of the system while processing multiple pulses enables the processing to separate signals based upon their Doppler frequency. Figure 3-1 shows the configuration for fully-adaptive STAP. Each array channel is followed by a tapped delay line filter that processes samples from multiple pulses received by the array. The outputs of the tapped delays are adaptively weighted and combined across all channels to form the residual output. The adaptive weights are selected based upon the received data allowing STAP to adjust its spatial-temporal response to adapt to changes in the signal environment.

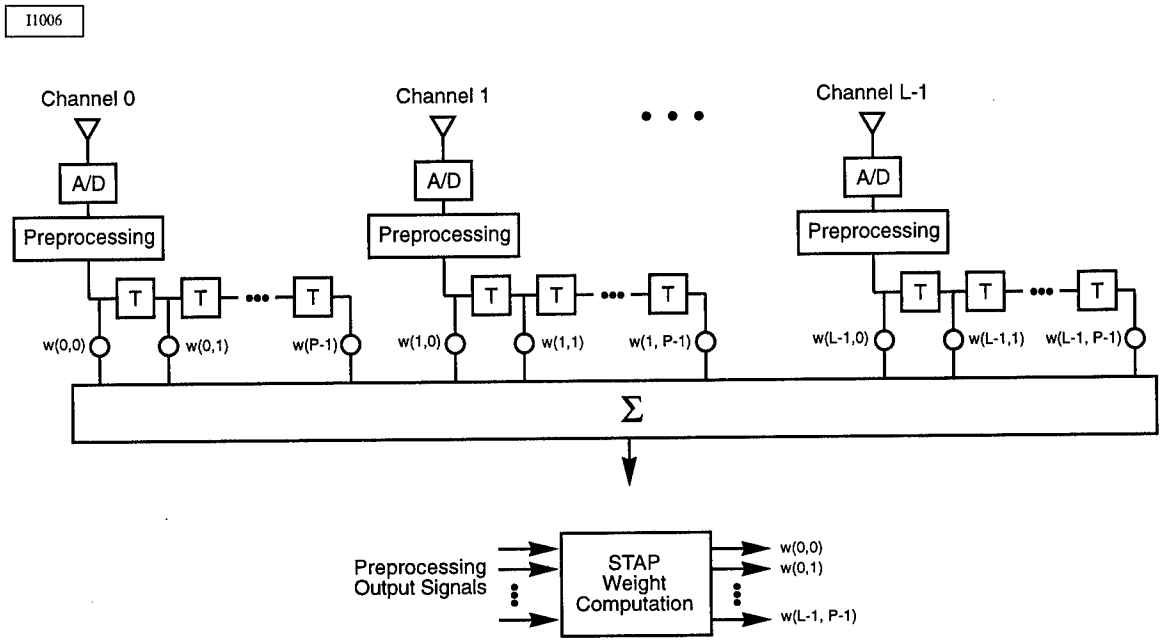

Figure 3-1. Fully-Adaptive STAP

For most airborne radar applications, implementation of a fully-adaptive STAP algorithm is not feasible due to the computational complexity of the weight computation process and the amount of data required to train the adaptive weights. As a result, sub-optimal, partially-adaptive techniques are used to implement the adaptive processing. A detailed taxonomy of partially-adaptive STAP algorithms having varying computational complexity and performance is provided in (Ward, 1994). The four classes of STAP algorithms are distinguished by the type of processing applied before adaptive processing. The four classes are: element-space pre-Doppler, element-space post-Doppler, beam-space pre-Doppler, and beam-space post-Doppler.

In this report, we focus on three element-space post-Doppler adaptive techniques that provide increasingly more effective clutter mitigation at the cost of higher processing throughput requirements. These algorithms are: post-Doppler adaptive DPCA, first-order Doppler-factored STAP, and higher-order Doppler-factored STAP. The algorithms compute a set of adaptive weights using a data domain approach that involves a matrix factorization

called the QR-decomposition (Golub and Van Loan, 1989, p. 211). The computational complexity of competing approaches is governed by the size of the QR-decomposition and the number of decompositions required.

In selecting an algorithm for the hard benchmark case, we considered two possible approaches: PRI-staggered post-Doppler STAP (Ward, 1994) and higher-order Doppler-factored STAP (DiPietro, 1992). Both algorithms were implemented in sequential C code and evaluated using MCARM data. We found that the eigenvalue spreads of the space-time covariance matrices used by the PRI-staggered approach were often very large due to the high correlation between noise samples of adjacent staggers, resulting in near-rank deficient covariance matrices. Consequently, we encountered substantial variations in the range-Doppler power values when the output of the double and single-precision implementations of this algorithm were compared. The conditioning problem could be reduced by using a strided stagger to decrease the noise correlation between adjacent staggers. Similar noise correlation effects were encountered using higher-order Doppler-factored processing techniques, however, the conditioning problem was significantly less than in the PRI-staggered case. Consequently, the `RT_STAP` benchmark provides a specification for third-order Doppler-factored STAP algorithm as the hard case. The architecture of a first-order Doppler-factored STAP algorithm is obviously derived from the more general higher-order approach. It provides a downward scaling in complexity while providing effective clutter and interference cancellation. The `RT_STAP` benchmark provides a specification for this algorithm as the medium case.

The post-Doppler adaptive DPCA algorithm forms the basis for our easy `RT_STAP` benchmark case. This algorithm was one of the first techniques developed to address the issue of Doppler-spread clutter in airborne radar. It can be shown that its architecture can also be derived from the more general PRI-staggered post-Doppler STAP algorithm. Both the performance and complexity of the post-Doppler adaptive DPCA algorithm are less than those of the first-order and third-order Doppler-factored algorithms. Due to its limited processing throughput requirements, the DPCA algorithm provides a practical solution to the clutter mitigation problem encountered in some airborne MTI applications. As a result, it has been applied in current radar systems.

The remainder of this section specifies the three benchmark cases. Section 3.1 first provides a functional specification for Doppler processing that forms a key component of all three post-Doppler adaptive algorithms. Section 3.2 specifies the higher-order Doppler-factored STAP algorithm and the general techniques used to compute the adaptive weights. Sections 3.3 and 3.4 specify the third- and first-order Doppler-factored STAP algorithms that correspond to the hard and medium benchmark cases, respectively. Finally, Section 3.5 specifies the post-Doppler adaptive DPCA algorithm that corresponds to the easy benchmark case.

3.1 Doppler Processing

The first component of all three post-Doppler adaptive algorithms is Doppler processing. It is implemented by applying a discrete Fourier transform of length K across P pulses of the preprocessed data for a given range cell and channel, where K represents the number of Doppler cells to be processed. A precomputed window function is applied to the data to reduce spectral leakage. The K complex data samples after Doppler processing can be written as:

$$x_5(l,k,r) = \sum_{p=0}^{P-1} d(p)\, x_4(l,p,r)\, e^{j(2\pi/K)pk}, \tag{3-1}$$

for $r = 0,\ldots,N_D - 1$, $k = 0,\ldots,K-1$, and $l = 0,\ldots,L-1$. In the above expression, the quantity $d(\cdot)$ represents the real-valued window function applied to the data samples, where $\vec{d}^{\,T}\vec{d}$ equals unity.

In practice, the discrete Fourier transform is implemented using an FFT. The data samples are zero padded, if necessary, so that the length of the FFT, denoted by K, is a power of two. A window function having length P is applied to the data before the transformation is performed. Application of the real-valued window function across all pulses of a given range cell and channel requires $2 \cdot P$ floating-point operations. Application of the FFT requires $5 \cdot K \cdot \log_2 K$ operations. It is left to the implementor to select an appropriate FFT algorithm. Therefore, a total of $5 \cdot K \cdot \log_2 K + 2 \cdot P$ operations is needed to implement Doppler processing for a given range cell and channel. The number of required

Doppler processing functional blocks depends on the number of range cells, channels, and type of adaptive processing algorithm. In general, the total number of computations required to implement Doppler processing is $L \cdot N_D \cdot (5 \cdot K \cdot \log_2 K + 2 \cdot P)$.

3.2 Higher-Order Doppler-Factored STAP

The higher-order Doppler-factored STAP algorithm can be one of the most effective STAP techniques known for clutter and interference suppression. Figure 3-2 shows the higher-order Doppler-factored STAP architecture of order Q. The architecture is composed of Doppler processing across all PRIs followed by adaptive filtering across sensors and adjacent Doppler bins. Adaptive filtering of the data uses simultaneous spatial and temporal degrees of freedom (DOF) in each specified Doppler bin. The spatial DOF are provided by the L array channels, while the temporal DOF are provided by the Q adjacent Doppler bins centered about the specified Doppler bin.

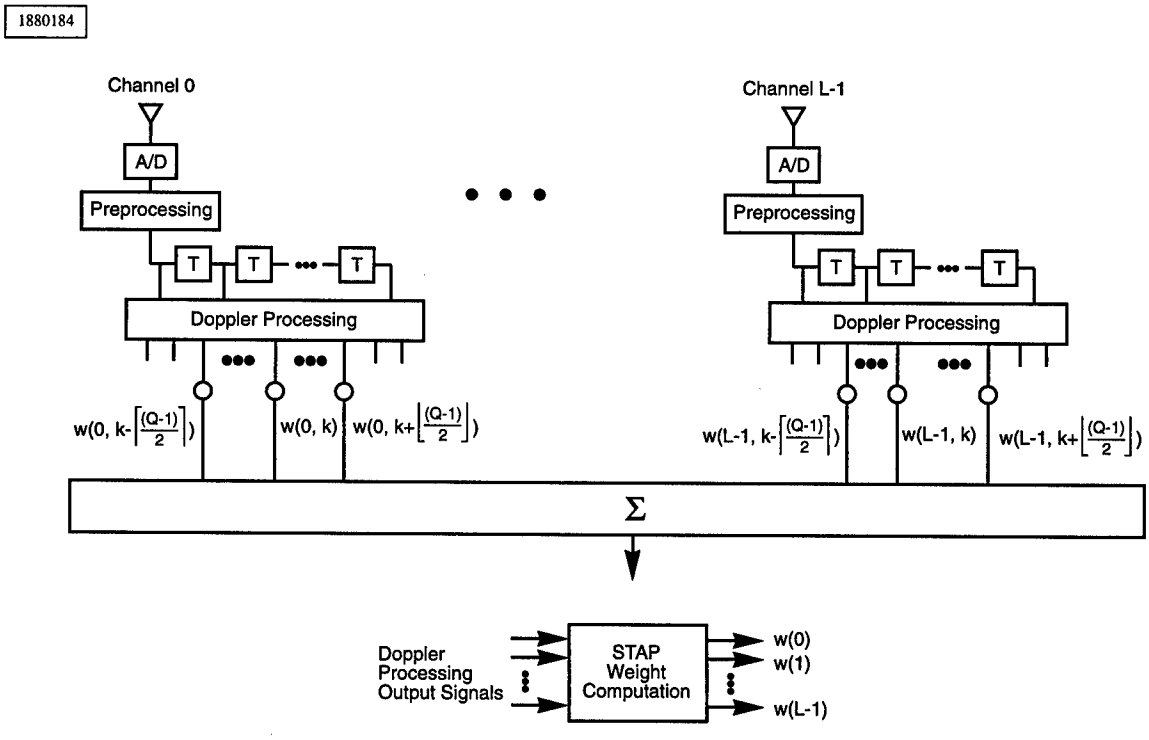

Figure 3-2. Higher-Order Doppler-Factored STAP

The adaptive weights for a particular range cell r and Doppler bin k are computed from the second-order statistics of the space-time snapshot vector $\vec{x}(k,r)$ consisting of data samples across the L array channels and the Q adjacent Doppler bins $k_{\min}$ through $k_{\max}$ that are centered about Doppler bin k. For Q-th order Doppler-factored STAP, we define $k_{\min}$ to be $\mathrm{mod}_K\left(k-\lfloor (Q-1)/2 \rfloor\right)$ and $k_{\max}$ to be $\mathrm{mod}_K\left(k+\lceil (Q-1)/2 \rceil\right)$, where $\mathrm{mod}_K(\cdot)$ is the modulo operator (e.g., $\mathrm{mod}_K(-1)=K-1$ and $\mathrm{mod}_K(K)=0$). The $\hat{L}\times 1$ space-time snapshot vector is defined to be:

$$\vec{x}(k,r)=\left[x_5(0,k_{\min},r)\cdots x_5(L-1,k_{\min},r)\cdots x_5(0,k_{\max},r)\cdots x_5(L-1,k_{\max},r)\right]^{\mathrm{T}}, \quad (3\text{-}2)$$

where $\hat{L}=L\cdot Q$ and $x_5(l,k,r)$ represents the data sample corresponding to the Doppler processing output for the r-th range cell, k-th Doppler bin, and l-th channel. Alternative bin selections are possible but are not relevant to this benchmarking effort. Given this definition of $\vec{x}(k,r)$, the $\hat{L}\times 1$ column vector of space-time adaptive weights $\vec{w}(k,r)$ is defined as

$$\vec{w}(k,r)=\left[w(0,k_{\min},r)\cdots\ w(L-1,k_{\min},r)\cdots w(0,k_{\max},r)\cdots w(L-1,k_{\max},r)\right]^{\mathrm{T}}, \quad (3\text{-}3)$$

where $w(l,k,r)$ represents the adaptive weight to be applied to the data sample corresponding to the r-th range cell, k-th Doppler bin, and l-th channel. The output of the adaptive processing for the k-th Doppler bin and r-th range cell is:

$$x_6(k,r)=\vec{w}^{\mathrm{H}}(k,r)\vec{x}(k,r). \quad (3\text{-}4)$$

The outputs $x_6(k,r)$ for $r=0,\ldots,N_{\mathrm{D}}-1$ and $k=0,\ldots,K-1$ result in the range-Doppler map following adaptive processing. Detection algorithms can then be applied to the result to locate targets in range and Doppler.

If we define the space-time covariance matrix to be:

$$\vec{\Psi}(k,r)=\mathrm{E}\left\{\vec{x}(k,r)\vec{x}^{\mathrm{H}}(k,r)\right\}, \quad (3\text{-}5)$$

where $E\{\cdot\}$ is the expectation operator, then the adaptive weights are obtained by solving the following system of linear equations (see pp. 326-330, Compton, 1988):

$$\vec{\Psi}(k,r)\vec{w}(k,r) = \gamma \vec{s}, \tag{3-6}$$

where γ is a scale factor chosen such that $\vec{w}^{\mathrm{H}}(k,r)\,\vec{s}$ equals unity. The $\hat{L}\times 1$ column vector $\vec{s}$ corresponds to the target space-time steering vector defined to be:

$$\vec{s} = \left[s(0,0)\cdots s(L-1,0)\cdots s(0,Q-1)\cdots s(L-1,Q-1)\right]^{\mathrm{T}}, \tag{3-7}$$

where

$$s(l,q) = \hat{s}(l)\,g\left(q + \mathrm{mod}_K\left(-\lfloor (Q-1)/2 \rfloor\right)\right), \tag{3-8}$$

$q = 0,\ldots,Q-1$, $l = 0,\ldots,L-1$, $\hat{s}(l)$ corresponds to the target spatial steering vector at the l-th channel,

$$g(k) = \sum_{p=0}^{P-1} d(p)\,e^{j(2\pi/K)pk}, \tag{3-9}$$

and $d(\cdot)$ represents the real-valued Doppler window. Note that the space-time steering vector is normalized such that $\vec{s}^{\mathrm{H}}\vec{s}$ equals unity.

From equation (3-6) we see that the higher-order Doppler-factored STAP algorithm clearly depends upon knowledge of the space-time covariance matrix. For most practical applications, this matrix is unknown and must be estimated from the data samples. In general, an estimate of the covariance matrix is computed by averaging over snapshot vectors from adjacent range cells. The precise training strategy for selecting the snapshot vectors that provides the optimal tradeoff of processing performance and computational complexity is, to some extent, an open question in the STAP community.

In this report, the training strategy involves dividing the range cells into M non-overlapping blocks containing N_R contiguous range samples, where $M = N_D / N_R$ and where N_R is selected so that the ratio is an integer. The covariance matrix for the k-th Doppler bin and m-th block of contiguous range cells is computed by averaging over the outer product of the snapshot vectors. That is:

$$\underline{\vec{\Psi}}(k,m) = \frac{1}{N_R} \sum_{r=r_1}^{r_1+N_R-1} \vec{x}(k,r)\vec{x}^H(k,r), \tag{3-10}$$

where $\underline{\vec{\Psi}}(k,m)$ is the estimate of the covariance matrix, $r_1 = mN_R$, $m = 0,...,M-1$, and $k = 0,...,K-1$. The estimate is used in place of the covariance matrix in equation (3-6) to compute the adaptive weight vector that is applied to all the data snapshots comprising the m-th block of range cells for the k-th Doppler bin.

If the $\hat{L} \times N_R$ space-time data matrix, $\vec{X}(k,m)$, is defined to be:

$$\vec{X}(k,m) = \left[\vec{x}(k,mN_R) \cdots \vec{x}(k,(m+1)N_R - 1)\right], \tag{3-11}$$

then

$$\underline{\vec{\Psi}}(k,m) = \frac{1}{N_R}\vec{X}(k,m)\vec{X}^H(k,m). \tag{3-12}$$

In practice, $\vec{X}(k,m)$ has full row rank equal to $\hat{L}$ because of background system noise. Equation (3-6) then becomes:

$$\vec{X}(k,m)\vec{X}^H(k,m)\underline{\vec{w}}(k,m) = \underline{\gamma} N_R \vec{s}, \tag{3-13}$$

where $\underline{\vec{w}}(k,m)$ corresponds to the weight vector applicable to all range cells in the m-th block and is computed using the data matrix $\vec{X}(k,m)$. The scale factor $\underline{\gamma}$ is chosen such that $\underline{\vec{w}}^H(k,m)\vec{s}$ equals unity.

The number of range samples used to estimate the covariance matrix should be at least twice the number of DOF (i.e., $\hat{L}$). Some authors suggest using between three to five times the number of DOF to minimize the loss in output signal-to-noise ratio that can occur when an estimate of the space-time covariance matrix is used. Note that the training strategy described above relies upon the knowledge of accurate spatial steering vectors to prevent significant target signal cancellation. More sophisticated training strategies exist that can alleviate the accuracy constraint; however, the application of these strategies can increase the computational complexity of the STAP algorithm by an order of magnitude.

3.2.1 Weights Computation

The weight vector is computed by first performing a QR-decomposition on the full column-rank space-time data matrix $\vec{X}^{\mathrm{T}}(k,m)$ defined in equation (3-11). Note that the transpose $\vec{X}^{\mathrm{T}}(k,m)$ is used to conform with the least-squares convention of having an over-determined system with more rows (N_{R}) than columns ($\hat{L}$). The QR-decomposition produces an $N_{\mathrm{R}} \times N_{\mathrm{R}}$ unitary matrix, $\vec{Q}$, and an $N_{\mathrm{R}} \times \hat{L}$ upper triangular matrix, $\vec{R}$, such that $\vec{X}^{\mathrm{T}}(k,m) = \vec{Q}\vec{R}$. The matrix $\vec{R}$ can be written as $\left[\vec{R}_1^{\mathrm{T}} \quad \vec{0}\right]^{\mathrm{T}}$, where $\vec{R}_1$ is a $\hat{L} \times \hat{L}$ full rank upper triangular matrix. The matrix product $\vec{X}(k,m)\vec{X}^{\mathrm{H}}(k,m)$ decomposes to:

$$\vec{X}(k,m)\vec{X}^{\mathrm{H}}(k,m) = \vec{R}^{\mathrm{T}}\vec{Q}^{\mathrm{T}}\vec{Q}^{*}\vec{R}^{*} = \vec{R}_1^{\mathrm{T}}\vec{R}_1^{*}, \tag{3-14}$$

where $\vec{Q}^{\mathrm{T}}\vec{Q}^{*} = \left(\vec{Q}^{\mathrm{H}}\vec{Q}\right)^{*} = I$. Since the matrix $\vec{Q}$ is not involved in the weight computation, it is not necessary to explicitly compute this matrix during the QR-decomposition process. A variety of methods exist for computing the QR-decomposition (Golub and Van Loan, 1989, pp. 211-220). We leave it to the developer to select an algorithm that provides the most efficient implementation for the target hardware architecture. If the modified Gram-Schmidt method is used (Golub and Van Loan, 1989, pp. 218-219), then $8 \cdot N_{\mathrm{R}} \cdot [L \cdot Q]^2$ floating-point operations are required to implement a single QR-decomposition corresponding to the m-th block of range cells for the k-th Doppler bin. A total of $K \cdot M \cdot \left(8 \cdot N_{\mathrm{R}} \cdot [L \cdot Q]^2\right)$ floating-point operations are required to implement all the QR-decompositions involved in the

application of Qth-order Doppler-factored STAP in each Doppler bin and each block of range cells in the data cube.

Following the QR-decomposition, forward elimination and backward substitution are performed to solve for the adaptive weights. The weight vector can be computed by first solving for the vector $\vec{p}$ in the expression:

$$\vec{R}_1^{\mathrm{T}} \vec{p} = N_{\mathrm{R}} \vec{s} \tag{3-15}$$

using forward elimination. The weight vector is determined by solving the expression:

$$\vec{R}_1^{*} \vec{r} = \vec{p} \tag{3-16}$$

using backward substitution. Algorithms for implementing both forward elimination and backward substitution are given in (Golub and Van Loan, 1989, pp. 87-88). The final weight vector $\underline{\vec{w}}(k,m)$ is equivalent to $\underline{\gamma}\vec{r}$, where $\underline{\gamma}$ is selected so that $\underline{\vec{w}}(k,m)^{\mathrm{H}} \vec{s} = 1$. As a result, $\underline{\gamma} = \left(\vec{r}^{\mathrm{H}} \vec{s}\right)^{-1}$.

Forward elimination and back substitution each require $4 \cdot [L \cdot Q]^2$ floating-point operations to implement. Clearly, implementation of the QR-decomposition dominates the computational complexity of the STAP weight computation process. Adaptive weights must be computed in each Doppler bin and each block of range cells. Consequently, the total number of computations required to solve for the adaptive weights for the entire data cube is $K \cdot M \cdot \left(8 \cdot [L \cdot Q]^2\right)$.

3.2.2 Weights Application

If we let $\underline{\vec{w}}(k,m)$ represent the adaptive weight vector to be applied to data corresponding to the k-th Doppler bin and the m-th block of range cells, then the N_{R} outputs of the STAP algorithm are given by the product of the data matrix and the weight vector:

$$\underline{\vec{w}}^{\mathrm{H}}(k,m) \vec{X}(k,m) . \tag{3-17}$$

This process requires $8 \cdot L \cdot Q \cdot N_R$ floating-point operations to implement, and it must be repeated in each Doppler bin and each block of range cells in the data cube. The total number of floating-point operations required to apply the adaptive weights to the data cube is $K \cdot M \cdot (8 \cdot L \cdot Q \cdot N_R)$.

3.3 Hard Benchmark Case: Third-Order Doppler-Factored STAP

The third-order Doppler-factored STAP algorithm can be one of the most effective STAP techniques known for clutter and interference suppression (DiPietro, 1992). This algorithm can provide a performance comparable to the fully-adaptive case while significantly reducing the computational complexity of the processing.

For the third-order Doppler-factored STAP processor, the space-time snapshot vector is formed using data from three adjacent Doppler bins centered around bin k. Computation counts for the hard benchmark case can be determined using the equations given in Section 3.2 with the number of sensors, L, set to 22 and the processing order, Q, set to 3.

3.4 Medium Benchmark Case: First-Order Doppler-Factored STAP

The first-order Doppler-factored STAP algorithm is one of the simplest post-Doppler STAP techniques known for clutter and interference suppression. In fact, since the technique does not adapt in the time domain (i.e., a single temporal DOF), it is not truly a STAP algorithm in the strict sense of this term. However, it is a spatially-adaptive algorithm that can provide substantial performance gains with moderate throughput requirements. Analysis results obtained through simulation and by processing data sets measured by the MCARM array and other sensor platforms indicate that the first-order Doppler-factored STAP algorithm that uses training sets of non-overlapping blocks of contiguous range samples is effective at nulling Doppler-spread clutter (Suresh Babu and Torres, 1995).

The architecture of the first-order Doppler-factored STAP algorithm corresponds to the processing of a single Doppler bin in the higher-order Doppler-factored architecture shown in

Figure 3-2. The computation counts for the medium benchmark case are readily obtained from the expressions given in Section 3.2 with the number of sensors, L, set to 16 and the processing order, Q, set to 1 (i.e., single temporal DOF).

3.5 Easy Benchmark Case: Post-Doppler Adaptive DPCA

The post-Doppler adaptive DPCA algorithm was one of the first techniques developed to address the issue of Doppler-spread clutter in airborne radar (Skolnik, 1990, pp. 16.8-16.14). It employs simultaneous spatial and temporal filtering, as shown in Figure 3-3, to suppress the sidelobe clutter competing with the target. The basic concept is to adjust the radar PRI such that the motion of the aircraft platform causes channel (i.e., phase center) 0 to move exactly into the spatial position of channel 1 after a single PRI (i.e., pulse). Consequently, the clutter signals at pulse p of channel 0 and pulse $p-1$ of channel 1 now appear to result from stationary ground scatterers and can be canceled using a simple two-pulse MTI filter. The processing differs from first-order Doppler-factored STAP only in that an additional time delay is included in channel one. This added time delay changes the Doppler processing implemented in the two channels so that the K complex data samples after Doppler processing are:

$$x_5(0,k,r) = \sum_{p=0}^{P-1} d(p)\, x_4(0,p,r)\, e^{j(2\pi/K)pk} \tag{3-18}$$

and

$$x_5(1,k,r) = \sum_{p=0}^{P-1} d(p)\, x_4(1,p+1,r)\, e^{j(2\pi/K)pk}, \tag{3-19}$$

for $r = 0,\ldots,N_D - 1$ and $k = 0,\ldots,K-1$. In other words, channel 0 performs Doppler processing on pulses 0 through $P-1$, while channel 1 processes pulses 1 through P. Note that channel 1 now requires that pulse P be preprocessed in lieu of pulse 0. The computation counts for the easy benchmark case are readily obtained from the expressions derived in Section 3.2 by using two channels (i.e., $L = 2$) and a processing order of one (i.e., $Q = 1$).

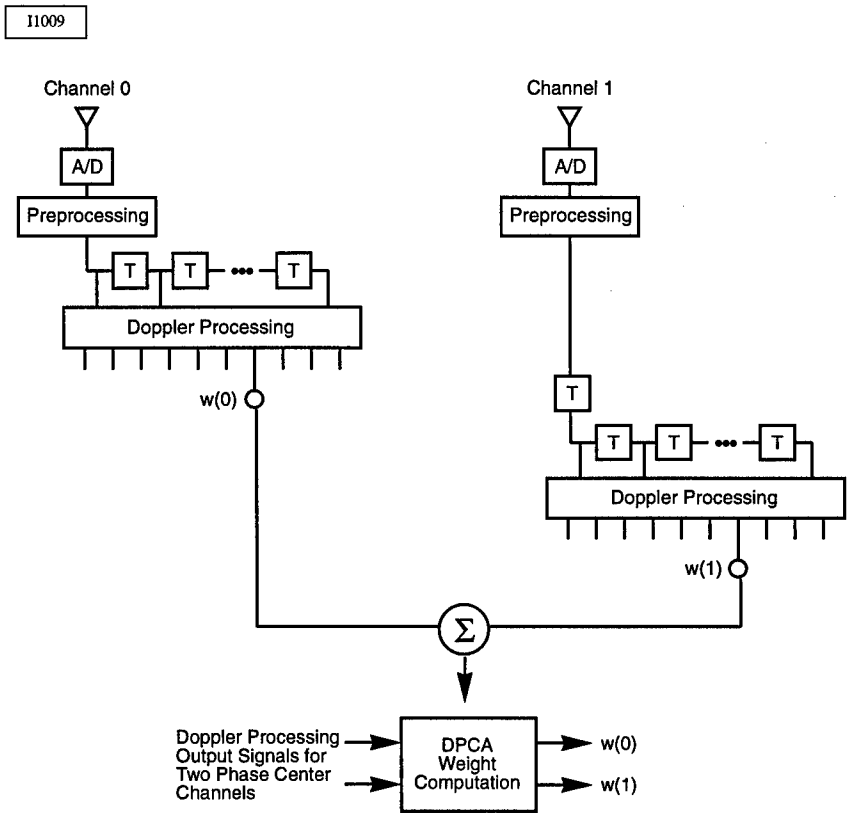

Figure 3-3. Post-Doppler Adaptive DPCA

3.6 Summary

In this section, we provided a functional specification for the three post-Doppler adaptive algorithms that form the hard, medium, and easy RT_STAP benchmark cases. These algorithms include the third-order Doppler-factored STAP algorithm for the hard case, the first-order Doppler-factored STAP algorithm for the medium case, and the post-Doppler adaptive DPCA algorithm for the easy case. Both the first-order Doppler-factored STAP algorithm and the post-Doppler adaptive DPCA algorithms implement a subset of the processing required by the higher-order Doppler-factored STAP algorithm resulting in a significant reduction in computational complexity. This reduction in complexity is largely obtained by reducing the size of the QR-decompositions required to compute the adaptive

weights. This section also evaluated the computational complexity of these methods. The results of this analysis are summarized in Table 3-1 for the higher-order Doppler-factored STAP algorithm. The computation counts for the hard, medium, and easy benchmark cases can be obtained by using ($L = 22$, $Q = 3$), ($L = 16$, $Q = 1$), and ($L = 2$, $Q = 1$), respectively.

Table 3-1. Higher-Order Doppler-Factored STAP Computation Counts

Function	Operation Count
Doppler Processing	$L \cdot N_D \cdot (5 \cdot K \cdot \log_2 K + 2 \cdot P)$
Weights Computation	$K \cdot M \cdot (8 \cdot [L \cdot Q]^2 \cdot (N_R + 1))$
Weights Application	$K \cdot M \cdot (8 \cdot L \cdot Q \cdot N_R)$

Definitions of the variables used in Table 3-1 are given below:

- L: Number of channels
- P: Number of pulses per Doppler processing block
- N_D: Number of samples per pulse after decimation
- K: Doppler FFT size (power of 2)
- M: Number of independent non-overlapping blocks N_D / N_R of contiguous range samples used to calculate the adaptive weights
- N_R: Number of contiguous range cells per weight computation
- Q: Processing order.

Section 4

MCARM Implementation

This section describes the timing specification and scalability study to be implemented as part of the RT_STAP benchmark. The parameters used to define both the timing specification and scalability study are based upon the MCARM data collection system. In Section 4.1 we briefly describe aspects of MCARM that are critical to the RT_STAP benchmark. Section 4.2 describes the timing specification and Section 4.3 outlines the scalability study.

4.1 MCARM Example

The MCARM (Multi-Channel Airborne Radar Measurements) system was developed in the early 1990s by Westinghouse for Rome Laboratory. The system is comprised of a 32 element antenna, transmitters, and receivers capable of generating L-band (1.3 GHz) radar measurements. The 32 antenna outputs are combined to form 22 channels. MCARM transmits a linear FM signal with a pulse repetition rate ranging between 250 and 2000 kHz and pulsewidth of 50 or 100 microseconds. The system also contains a recording system that enables the operator to record up to a 100 millisecond CPI data block from all 22 channels, at a rate of 5 MHz. The data recording system can store a single CPI every 2 seconds. The system is mounted on a BAC 1-11 aircraft and can be used to collect airborne radar data suitable for evaluating the performance of STAP algorithms. A number of data collection experiments have been conducted in the Chesapeake Bay area and several data sets have been made available by Rome Laboratory (see `http://sunrise.oc.rl.af.mil`).

Table 4-1 summarizes the parameter values used to determine timing requirements and evaluate the throughput for both the preprocessor and STAP algorithm when applied to the MCARM example. Note that the MCARM array provides IF data consisting of real integer A/D samples generated with a sampling rate of 5 MHz. After conversion to baseband, the I/Q is decimated by a factor of four to reduce the sample rate to 1.25 MHz.

Table 4-1. Parameter Values for the Three Benchmark Cases

Parameter	Value
Number of channels to be processed (L)	see Note 1
Number of pulses per Doppler processing block (P)	64
Number of channels in binary file containing input data cube	22
Number of PRIs in binary file containing input data cube	65
Number of samples per pulse before decimation (N)	1920
Decimation factor (D)	4
Number of samples per pulse after decimation (N_D)	480
FIR filter length used in video-to-I/Q conversion (K_a)	36
FIR filter length used in array calibration (K_c)	3
FIR filter length used in pulse compression (K_p)	63
Convolution length used to implement calibration and pulse compression ($\tilde{R}$)	192
FFT size used by the overlap-save method (R)	256
Number of blocks used to implement the overlap-save method (B)	3
Doppler FFT size (K)	64
Number of independent non-overlapping range blocks (M)	see Note 2
Number of range cells per weight computation (N_R)	see Note 3
Processing Order (Q)	see Note 4

Note 1: L for the hard, medium, and easy benchmark cases is 22, 16, and 2, respectively.

Note 2: M for the hard, medium, and easy benchmark cases is 2, 6, and 6, respectively.

Note 3: N_R for the hard, medium, and easy benchmark cases is 240, 80, and 80, respectively.

Note 4: Q for the hard, medium, and easy benchmark cases is 3, 1, and 1, respectively.

For the evaluation of the hard, medium, and easy benchmark cases, we used 22, 16, and 2 of the 22 available MCARM data collection channels, respectively. For all three cases, the CPI consisted of 64 contiguous pulses. These parameters were selected to obtain the operation rates that met the requirements of the benchmark cases. The high performance computer must input 0.49, 3.93, and 5.41 Mbytes of short integer data per CPI for the easy, medium, and hard benchmark cases, respectively.

4.2 Timing Specification

A key element of any real-time benchmark is the specification of the period and latency required by the application. The period is defined to be the time between input data sets while latency is the time required to process a single data set. The latency corresponds to the time from when the first data leaves the data source to the time the final result is output to the data sink. The strictness of the latency requirement determines the difficulty, and in some cases even the feasibility, of the parallel implementation.

For RT_STAP both the period and latency are closely associated with the CPI of the radar system. We assume data is measured continuously and must be processed independently from one CPI to another. As a result, the period corresponds to a single CPI. For the MCARM example used in Section 4.1, the period equals 32.25 milliseconds corresponding to a CPI with 64 pulses.

We define two latency cases for the RT_STAP benchmark: short and unlimited. For the unlimited latency case, we make no restrictions on the length of time required to complete the processing of the data cube associated with a single CPI. The implementor is encouraged to minimize processor size without regard to latency. This result provides a lower bound on the size of the processor required to meet the period requirement of the benchmark. The short latency case requires that the processor input the CPI data cube from the source, process the data cube, and output the results to the data sink within 5 CPIs. This corresponds to a latency of 161.25 milliseconds for the MCARM example. One of the considerations in this choice is to force a parallel implementation of the QR-decomposition for the hard benchmark case.

4.3 Scalability Study

Conventional benchmarking approaches evaluate scalability by varying the size of the processing problem (typically the amount of data to be processed) while holding the period and latency fixed. Although the amount of data increases as we increase the difficulty of the benchmark, RT_STAP evaluates the scalability of the architecture based largely upon increasing algorithm complexity for a fixed period and latency. The subsequent discussion evaluates the operation rate required to implement both the preprocessing described in Section 2 and the post-Doppler adaptive algorithms described in Section 3 for the MCARM data collection system. The results described in this section correspond to three data points of the scalability study associated with the easy (DPCA), medium (first-order Doppler-factored STAP), and hard (third-order Doppler-factored STAP) benchmark cases.

Table 4-2 gives the computational throughput requirements for both the preprocessing described in Section 2 and the post-Doppler adaptive algorithms described in Section 3 based upon the MCARM parameters described in Table 4-1. The operation rates are specified in billions of floating-point operations per second (Gflop/s) and are computed by dividing the operation counts given in Sections 2 and 3 by the period. For this scenario, the period is equivalent to the duration of the CPI and is 32.25 milliseconds.

For the easy, medium, and hard benchmark cases, the preprocessing and the post-Doppler STAP algorithms require a total operation rate of 0.60, 6.46, and 39.81 Gflop/s, respectively. For the easy case, preprocessing dominates the computation complexity, accounting for over 82 percent of the processing operation rate. For the medium benchmark case, first-order Doppler-factored STAP accounts for slightly less than 40 percent of the processing required for the 16 channel data cube (i.e., 2.59 Gflop/s for the STAP algorithm versus 3.87 Gflop/s for the preprocessing). For the hard benchmark case, the computational complexity of the STAP algorithm significantly increases and accounts for nearly 87 percent of the processing required for the 22 channel data cube (i.e., 34.50 Gflop/s for the STAP algorithm versus 5.31 Gflop/s for the preprocessing). Table 4-3 lists the number and size of the QR-decompositions required to compute the adaptive weights for each benchmark case.

Table 4-2. Operation Rates for the Three Benchmark Cases

Function	Operation Rate (Gflop/s)		
	DPCA	First-Order Doppler-Factored STAP	Third-Order Doppler-Factored STAP
Video-to-I/Q Conversion	0.22	1.77	2.43
Calibration and Pulse Compression	0.26	2.10	2.88
Preprocessing Total	0.49	3.87	5.31
Doppler Processing	0.06	0.49	0.67
Weights Computation	0.03	1.98	33.33
Weights Application	0.02	0.12	0.50
Adaptive Processing Total	0.11	2.59	34.50
Total	0.60	6.46	39.81

Table 4-3. Number and Size of QR-Decompositions for the Three Benchmark Cases

Benchmark	# of QRs	Matrix Size
Easy	384	80 x 2
Medium	384	80 x 16
Hard	128	240 x 66

Section 5

Sequential Software Implementation

A sequential software implementation of the functional specification was developed for the preprocessing and three post-Doppler adaptive algorithms that make up the easy, medium, and hard benchmarks described in Sections 2 and 3. This implementation consists of 1600 lines of sequential C source code designed to run on both Sun and Silicon Graphics workstations under the UNIX-based Solaris 2.3 (SunOS 5.3) and IRIX 5.3 operating systems, respectively. The software is intended to support validation of parallel implementations of the functional specification described in this report and may be used as a starting point for a parallel implementation of the STAP algorithm. The code is not intended to provide an efficient parallel implementation for any particular high performance computing system.

Figure 5-1 illustrates the sequential processing of a single real input data cube. The software decomposes into a series of processing functions and corner-turns. Each cube shown in Figure 5-1 depicts the ordering of the data with the rows of a face corresponding to the fastest changing variable, columns are indexed second and each face indexed at the slowest rate. For example, in the first data cube shown in Figure 5-1, time is the fastest running variable, PRI (i.e., pulse) is second, and channel is third. Each arrow in the figure corresponds to a function applied to the data. These functions correspond to either a data processing step or corner-turn. The software is structured so that each process is explicitly implemented by a particular code segment.

At various points in the functional specification, selection of a particular algorithm for implementing key components of the preprocessing and STAP algorithms was left to the software developer. In particular, selection of FFT algorithms and methods for implementing the QR-decomposition were left open. In the development of the sequential software, all FFTs were implemented using the algorithm defined in (Press, *et al.*, 1988, pp. 411-412) and the modified Gram-Schmidt QR-decomposition technique described in (Golub and Van Loan, 1989, pp. 218-219) was used to compute the adaptive weights.

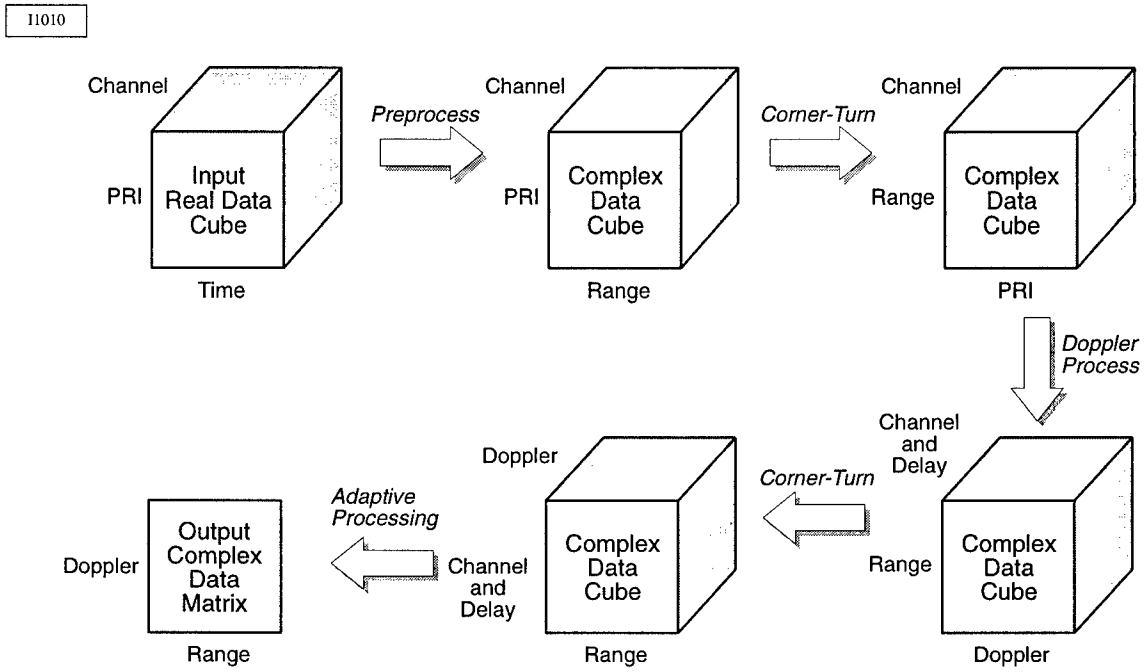

Figure 5-1. Sequential Software Implementation

The software has been used to process a limited number of MCARM data cubes. With some restrictions, the C source code, calibration coefficients, spatial steering vector, FIR lowpass filter coefficients, test input data cube, and post-Doppler adaptive processing results can be obtained directly from Ron Williams whose electronic mail address is `ronw@mitre.org`. The results obtained by applying the sequential software implementing non-adaptive beamforming processing and first-order Doppler-factored STAP to this data cube are shown in Figures 5-2 and 5-3, respectively, and are described below. It should be noted that the test input data cube is a subset of an original MCARM data cube, where the first 600 time samples for each PRI of each channel have been eliminated since they contain the recorded transmitted pulse.

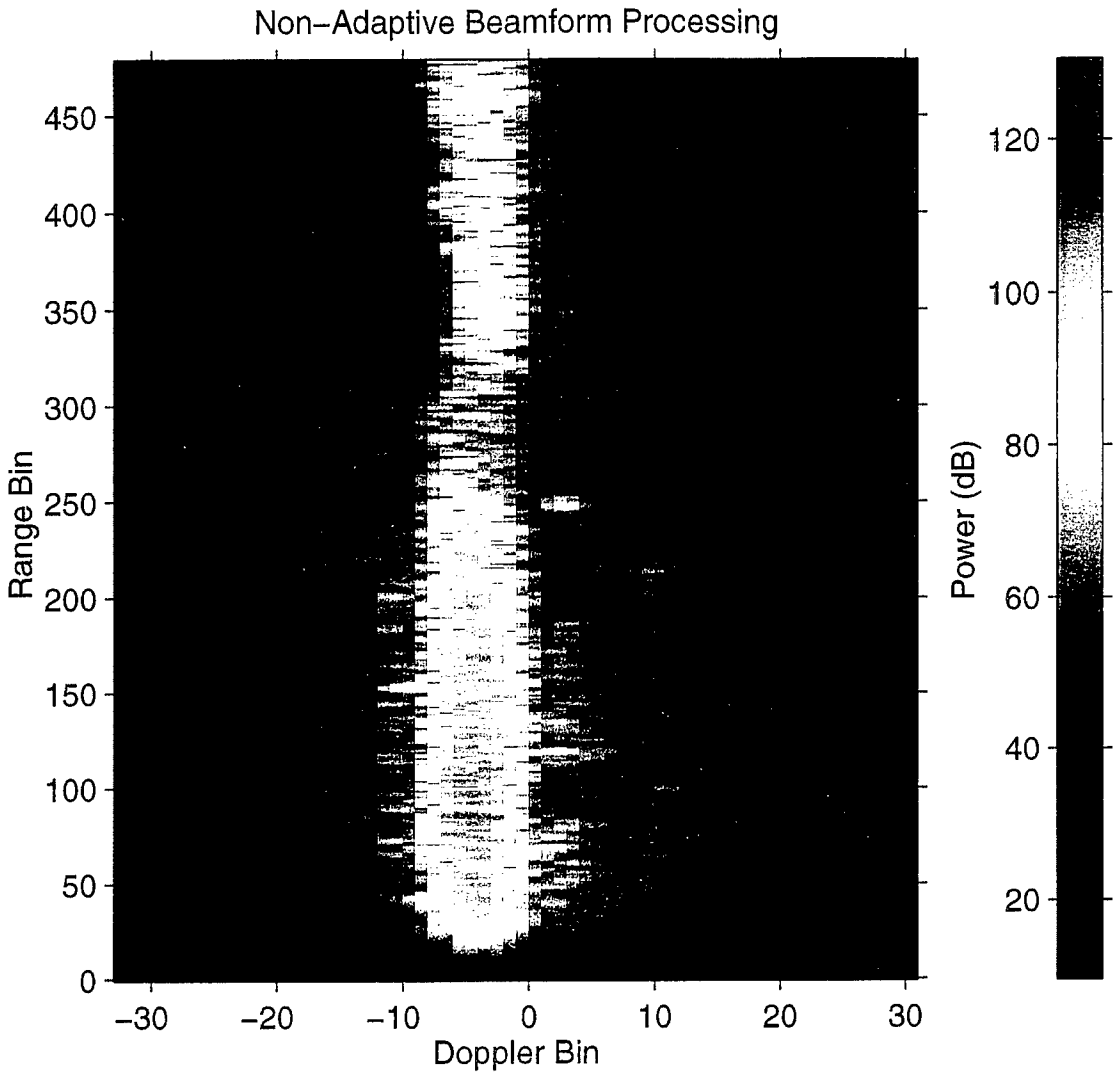

Figure 5.2 Range-Doppler Map Following Non-Adaptive Beamforming Processing

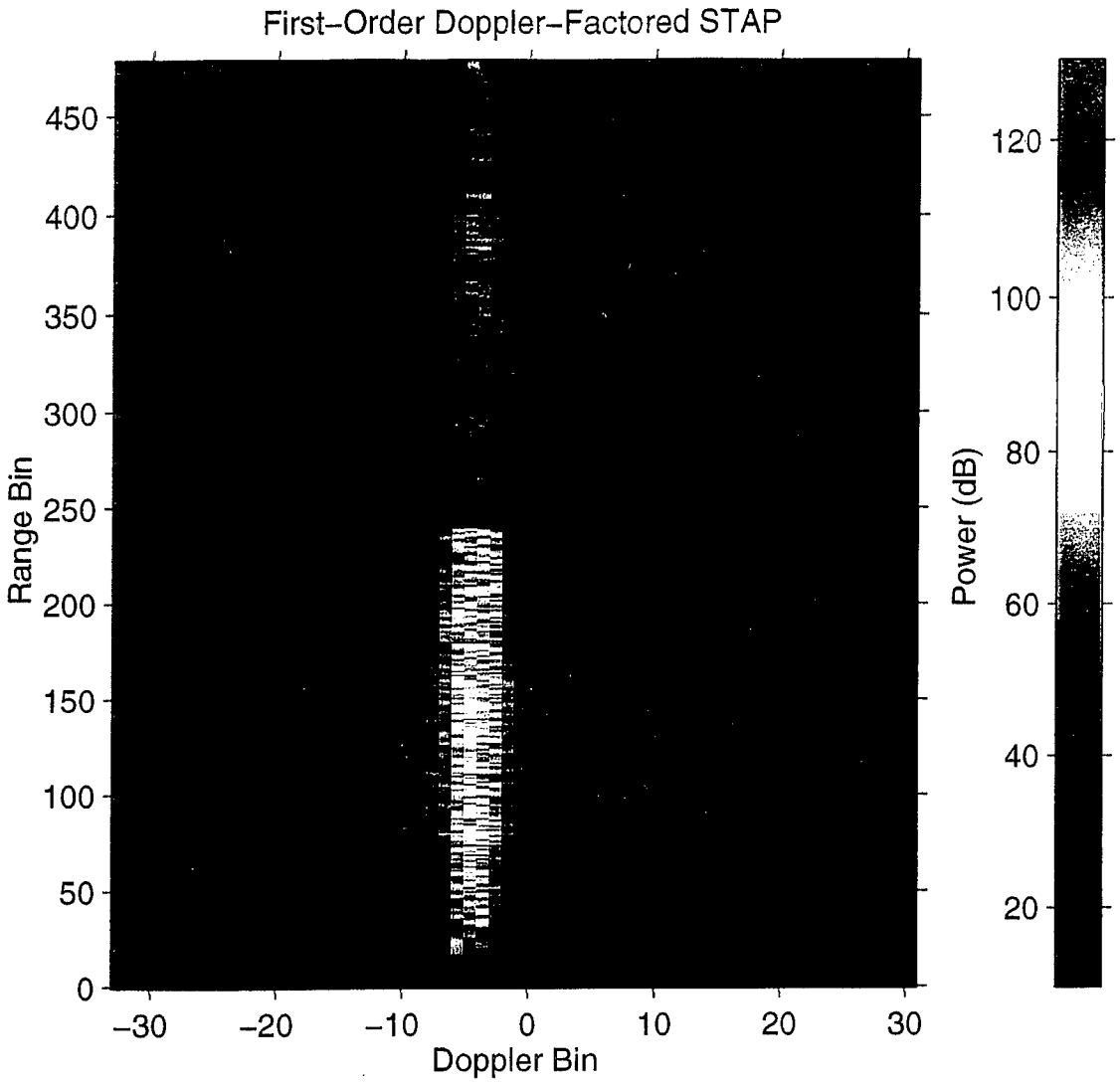

Figure 5.3 Range-Doppler Map Following First-Order Doppler-Factored STAP

5.1 Building the Software

The software is delivered on an 8 mm tape that contains both the source code and `Makefile` used to create the executable code. The code can be removed from the tape by typing `tar -xvf <device>` at the prompt, where `<device>` corresponds to the tape drive being used to extract the data (e.g., `/dev/rst0`). The directory `MITRE-RT_STAP` will be created that contains the source code and data files necessary to run the sequential code.

The source code is compiled by first moving to the directory `MITRE-RT_STAP` and successively typing the commands `make clean`, `make depend` and `make` at the prompt. It is assumed that an `ANSI C` compiler is available and that the `PATH` environment variable is properly set to provide access to this compiler. For example, in our system architecture we used the `gcc` compiler, version 2.7.2, with the `-ansi` and `-pedantic` compiler switches. A compile-time switch within the `Makefile` also provides a choice between a single or double-precision version of the sequential software. The user can simply choose between either `PRECISION=DOUBLE` or `PRECISION=SINGLE` in the `Makefile`.

5.2 Running the Software

The software is run by typing `RT_STAP PARAMETERS` at the prompt. The ASCII input parameters file (`PARAMETERS`) and the remaining three ASCII data files (`IQ_filt.dat`, `calib_filt.dat`, and `steering_vec.dat`) required by the software are provided on the tape. The contents of these files, along with descriptions of the other input parameters, are provided in Table 5-1. The real input test data cube is contained in the short integer binary file `input_cube.bin`. The post-Doppler adaptive processing range-Doppler complex outputs calculated using the double-precision version of the sequential software are also provided on the tape. Specifically, the double-precision complex binary files `output_hard.bin.VAL`, `output_medium.bin.VAL`, and `output_easy.bin.VAL` correspond to third-order Doppler-factored STAP, first-order Doppler-factored STAP, and post-Doppler adaptive DPCA, respectively. The non-adaptive beamform processing range-Doppler outputs, calculated by using a processing order

Table 5-1. Sequential Software Input Parameters in File PARAMETERS

Name	Description	Value
iL	Number of channels to be processed	see Note 1
iLFile	Number of channels in file containing input data cube	22
iLFileOffset	First channel in data file to be processed	see Note 2
iP	Number of pulses per Doppler processing block	64
iPFile	Number of PRIs in file containing input data cube	132
iPFileOffset	First PRI in data file to be processed	0
iN	Number of samples per pulse at A/D rate before decimation	1920
df0	Ratio of center frequency to A/D sampling rate	0.25
iM1	FIR filter length used in video-to-I/Q conversion	36
dBs	Bandwidth of transmitted linear FM signal	1.0E06
iD	Sampling rate decimation factor	4
dfAD	A/D sampling rate	5.0E06
iM2	FIR filter length used in array calibration	3
dTau	Uncompressed pulsewidth of transmitted linear FM signal	50.4E-06
iR_fftlen	FFT size used by overlap-save fast convolution	256
iWinDoppler	Type of window taper used in Doppler processing (1=rectangular, 2=Hanning, 3=Hamming, 4=Blackman)	4
iWinPCF	Type of window taper used in pulse compression (1=rectangular, 2=Hanning, 3=Hamming, 4=Blackman)	4
FIR_file	Name of ASCII input file containing FIR filter coefficients used in video-to-I/Q conversion	IQ_filt.dat
ChanBalFile	Name of ASCII input file containing FIR filter coefficients used in array calibration	calib_filt.dat
InputDataFile	Name of binary file containing input data cube	input_cube.bin

Note 1: iL for the hard, medium, and easy benchmark cases is 22, 16, and 2, respectively.

Note 2: iLFileOffset for the hard, medium, and easy benchmark cases is 0, 2, and 2, respectively.

Table 5-1. Sequential Software Input Parameters in File PARAMETERS (Concluded)

Name	Description	Value
I_Adapt	Type of post-Doppler processing (0=non-adaptive, 1=DPCA, 2=STAP)	see Note 3
iQ	Post-Doppler processing order	see Note 4
iDeltaR	Number of range cells used to compute adaptive weights	see Note 5
iRmin	Minimum range index to be post-Doppler processed	0
iRmax	Maximum range index to be post-Doppler processed	479
SteerVectorFile	Name of ASCII input file containing spatial steering vector	steering_vec.dat
FinalDataFile	Name of binary file containing output range-Doppler data matrix	see Note 6
ValDataFile	Name of binary file containing double-precision output range-Doppler data matrix used for validation	see Note 7

Note 3: I_Adapt for the hard, medium, and easy benchmark cases is 2, 2, and 1, respectively.

Note 4: iQ for the hard, medium, and easy benchmark cases is 3, 1, and 1, respectively.

Note 5: iDeltaR for the hard, medium, and easy benchmark cases is 240, 80, and 80, respectively.

Note 6: FinalDataFile for the hard, medium, and easy benchmark cases is ouput_hard.bin, output_medium.bin, and output_easy.bin, respectively.

Note 7: ValDataFile for the hard, medium, and easy benchmark cases is ouput_hard.bin.VAL, output_medium.bin.VAL, and output_easy.bin.VAL, respectively.

of one and setting the weights equal to the normalized space-time steering vector are also provided in the double-precision complex binary file output_nonadapt.bin.VAL. The corresponding range-Doppler maps for non-adaptive beamform processing and first-order Doppler-factored STAP are shown in Figures 5-2 and 5-3, respectively.

The program generates timing measurements for the sequential software and sends them to standard output. Wall clock time for several major functional blocks are generated, including: disk I/O, preprocessor, corner turns, Doppler processing, the STAP algorithm, various miscellaneous functions, and finally a total processing time. The timing facility used in the sequential software is the BSD compatible gettimeofday() function.

Section 6

Implementation and Reporting Guidelines

This section provides a set of implementation and reporting guidelines for distributed memory multicomputers—the initial intended target of the RT_STAP benchmark. The target multicomputers contain processing nodes with one or more general-purpose microprocessors or digital signal processing chips that support floating-point operations. These guidelines require the use of the "test bench" configuration described in (Brown, *et al.*, 1995) and (Games, 1996) to measure real-time performance of the benchmark implementation under steady-state conditions. The test bench requires data sources and data sinks that are dedicated to providing inputs to the function under test and storing results. These sources and sinks may be located on the machine under test so that an actual I/O interface need not be implemented. In (Games, 1996), a set of general guidelines for implementing compact applications that are based upon the use of the test bench has been reported. We have adapted these guidelines to the RT_STAP benchmark.

The RT_STAP benchmark guidelines require that:

1. The input data cube must be stored originally on a source node(s) in memory that is not directly associated with the processors that implement the RT_STAP preprocessing or STAP functions. The data cube has dimensions of channels, pulses, and time. Data must be stored in a single contiguous set of memory locations with time samples grouped first as a function of pulses and then channels. If the data block is too large to be stored in a single source node, then it should be partitioned by channels.

2. When establishing timing performance the same data cube can be repeatedly input to the processors that implement the RT_STAP function (to avoid the need for disk I/O).

3. The results must be output to a sink node(s) and stored in memory not directly associated with the processors that implement the RT_STAP preprocessing or STAP functions. The output range-Doppler data matrix must be stored in a single contiguous

set of memory locations with the range samples grouped as a function of the Doppler frequency, or vice versa.

4. The source and sink nodes may be implemented on the same or different processing nodes.

5. The processing latency for a problem instance is measured as follows. A time stamp t_s is calculated at the data source right before the first input data for this instance is sent from the source node. A second time stamp t_c is calculated at the data sink right after the final corresponding results are received by the sink node. The processing latency for this problem instance is then $t_c - t_s$. This requires a synchronized global clock if the source and sink are on physically separated nodes. Period measurements are calculated as the difference of successive values of t_c corresponding to successive problem instances. Latency and period measurements can be calculated off-line from the time stamp data. Care must be taken that data collection does not cause any unintended disk I/O to occur during a run.

6. In the case that the input data cube and output range-Doppler data matrix do not fit in the memory of a single node, then multiple source and sink nodes are necessary. The time stamp t_s of a problem instance should occur before any data is sent from the multiple sources. The processing of that instant is considered completed when all sink nodes have received all their results.

7. A benchmark run to establish valid timing performance should run continuously for at least 15 minutes to account for any operating system dropout problems.

8. The following information and statistics should be calculated during a post processing stage for a single benchmark run:

    a. Histogram of period measurements. Maximum, average, and minimum period. The benchmark is considered valid only if the maximum period observed is less than or equal to the period specification. This must be repeatable.

b. Histogram of latency measurements. Maximum, average, and minimum latency. The benchmark is considered valid only if the maximum latency observed is less than or equal to the latency specification. This must be repeatable.

c. Some small number of initial problem instances can be ignored to eliminate start-up anomalies, if present. If this is done then the number of ignored instances should be stated.

9. For each latency case and for each feasible problem size, the smallest machine that produces a valid benchmark result should be determined. Machine size is measured in terms of the number of processing nodes used, not including processing nodes used to implement the source and sink. The only exception to this rule is that any "excess" source and sink nodes beyond the minimum number that are required to store one copy of the input data cube and output matrix should also be counted in the machine size. Standard commercial-off-the-shelf hardware and system software configurations should be used. If a machine supports multiple configurations (for example, different amounts of memory at the processing nodes), then these different configurations must be itemized and benchmarked separately.

10. The maximum period for a benchmark run is used to report the sustained Gflop/s operation rate of the implementation. The operation rate is defined to be the total floating-point operation count determined using the results presented in Section 2 for the preprocessing and Section 3 for the STAP algorithm divided by the maximum period measurement. Alternatively, this number can be obtained by multiplying the operation rate from Table 4-2 by the required period and dividing by the maximum period measured. This value should be divided by the theoretical peak processing rate of the size machine used in the processing to determine the processing utilization percentage.

11. The following scalability information should be generated for each latency case:

a. Minimum machine size as a function of the benchmark case (easy, medium, hard).

b. Processing utilization percentage as a function of the benchmark case (easy, medium, hard).

12. Validation criterion: It would be desirable to define an application-specific validation criterion associated with STAP system performance (e.g., detectability of a weak target), and to let basic issues such as number representation be determined by the implementor. Such a point-of-view is beyond the scope of this initial effort. Instead the expectation is that the RT_STAP benchmark will be implemented initially in 32-bit single-precision floating point, and the validation criterion was chosen to reflect this fact. We do not preclude the use of alternative number representations (e.g., fixed-point accelerators to implement the preprocessing), but the validation criterion may need to be adjusted in this case.

Differences between the results of the parallel and sequential implementations should be due to rounding error and, as a result, they should be relatively small. The validation process compares the range-Doppler results generated by the single-precision parallel software implementation with the double-precision results generated using the sequential software. All calculations performed as part of the validation process should be calculated using double-precision values. Let $z(k,r)$ be the processed radar return associated with Doppler bin k and range cell r generated using the double-precision version of the sequential software, and let $\hat{z}(k,r)$ be the radar return computed using a single-precision parallel software implementation. The validation criterion requires that the peak change in power values (measured in dB) be less than 1 dB for power values (single or double-precision) greater than a specified threshold (i.e., 10 dB below the estimated noise floor P_N). Specifically, the validation criterion is given by:

$$\max_{k,r} \left| 10\log_{10}|z(k,r)|^2 - 10\log_{10}|\hat{z}(k,r)|^2 \right| < 1,$$

where the maximum is over all values k and r such that $10\log_{10}|z(k,r)|^2 > (P_N - 10)$ or $10\log_{10}|\hat{z}(k,r)|^2 > (P_N - 10)$. Note that this validation criterion has an equivalent representation using a definition of relative error:

$$\max_{k,r} 10\log_{10}\left(\frac{\left||z(k,r)|^2 - |\hat{z}(k,r)|^2\right|}{|z(k,r)|^2}\right) < -6.9.$$

The noise floor is estimated over a region in the range-Doppler map that is removed from the mainlobe clutter. For this benchmarking effort, P_N is estimated as

$$P_N = 10\log_{10}\left(\frac{1}{\lfloor 0.2N_D \rfloor \cdot \lfloor 0.2K \rfloor} \sum_{k=\lceil 0.3K \rceil}^{0.5K-1} \sum_{r=\lceil 0.8N_D \rceil}^{N_D-1} |z(k,r)|^2\right).$$

The post-Doppler adaptive processing range-Doppler results $z(k,r)$ calculated using the double-precision version of the sequential software are provided on the tape, as discussed previously.

Section 7

Conclusion

This report provides a specification of the RT_STAP benchmark for evaluating the application of high performance computers to the real-time implementation of space-time adaptive processing techniques on embedded platforms. The scalability study outlined in the RT_STAP benchmark varies the sophistication and computational complexity of algorithms to be implemented along with the size of the data processing problem. This report provided functional specifications for three post-Doppler adaptive algorithms corresponding to the easy, medium, and hard cases of the benchmark. These algorithms include: DPCA, first-order Doppler-factored STAP, and third-order Doppler-factored STAP algorithms. These algorithms were selected due to the range of performance and computational complexity they provide.

Preprocessing that is typically applied to the data samples collected by an antenna array used in airborne surveillance applications was described in Section 2. The preprocessing implements several critical functions including video-to-I/Q conversion, array calibration, and pulse compression. These standard signal processing functions can represent a significant component of the processing chain required for implementing post-Doppler adaptive processing. Although usually implemented with special-purpose hardware, we view preprocessing as a potential application of HPC technology. By using HPC technology to implement preprocessing, the number of hardware interfaces required to implement an MTI radar can be reduced thus minimizing the complexity of the system and simplifying future algorithm upgrades.

A key component of the RT_STAP benchmark is the specification of the period and latency requirements. For this benchmark we used the CPI of the MCARM data collection system to specify the period and latency. The period corresponds to the duration of a single CPI and is equal to 32.25 milliseconds. In the benchmark we specified two latency cases: long and short. The long latency case makes no restriction on the latency of the implementation and provides a lower bound on processor size required to meet the

throughput requirement determined by the period. The short latency case requires a solution to be computed in less than 161.25 milliseconds corresponding to 5 CPIs of the MCARM data collections system.

In this report, an analysis of the computational throughput of both the preprocessing and the STAP algorithms was conducted for the MCARM data collection system. We found that the operation rate required to implement the three benchmark cases including both preprocessing and adaptive processing was: 0.60 Gflop/s for the easy case (DPCA), 6.46 Gflop/s for medium benchmark case (first-order Doppler-factored STAP), and 39.81 Gflop/s for the hard case (third-order Doppler-factored STAP). The DPCA algorithm can be easily implemented using current HPC technology. While implementation of the medium benchmark is not trivial, it can likely be achieved on a large number of HPC platforms. The third-order Doppler-factored STAP algorithm, while representing a likely migration path for future STAP-based MTI radars, presents a challenging benchmark for most HPC systems.

As our analysis shows, the QR-decomposition used to compute the adaptive weights is the key contributor to the computational complexity of the three algorithms. Variations in computational complexity are largely due to the number and size of the QR-decompositions used to implement the weight computation process for each algorithm. For the short latency requirement, it is expected that the easy and medium benchmark cases will not require the parallel implementation of the QR-decomposition. However, the short latency requirement is expected to force a parallel implementation for the hard benchmark case, thus significantly increasing the complexity of implementing this algorithm. The performance of a single processor implementation of the QR-decomposition is a key enabler for efficient STAP implementations. The single-processor QR-decomposition timings for matrix sizes representative of real STAP applications should be tracked as technology improves and is the focus of the single-processor QR-decomposition benchmark introduced in Appendix B.

Finally, in Sections 5 and 6 of this report we describe a sequential implementation of the three algorithms and implementation and reporting guidelines for the `RT_STAP` benchmark. The sequential C code can be used to validate parallel implementations of the benchmark.

List of References

Brennan, L., and I. S. Reed, 1973, "Theory of Adaptive Radar," *IEEE Transactions on Aerospace and Electronic Systems,* Vol. 9, pp. 237-252.

Brown, C. P., R. A. Games, and J. J. Vaccaro, 1995, *Real-Time Parallel Software Design Case Study: Implementation of the RASSP SAR Benchmark on the Intel Paragon*, MTR 95B0000095, The MITRE Corporation, Bedford, MA.
(Available at http://www.mitre.org/research/hpc)

DiPietro, R. C., 1992, "Extended Factored Space-Time Processing for Airborne Radar Systems," *Twenty-Sixth Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, California, pp. 425-430.

Eaves, J. L., and E. K. Reedy, 1987, *Principles of Modern Radar,* New York: Van Nostrand Reinhold.

Games, R. A., 1996, *Benchmark Methodology for Real-Time Embedded Scalable High Performance Computing*, MTR 96B0000010, The MITRE Corporation, Bedford, MA.
(Available at http://www.mitre.org/research/hpc)

Golub, G. H., and C. F. Van Loan, 1989, *Matrix Computations 2nd Edition,* Baltimore, MD: The Johns Hopkins University Press.

Oppenheim, A. V., and R. W. Schafer, 1975, *Digital Signal Processing,* Englewood Cliffs, NJ: Prentice-Hall Inc.

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, 1988, *Numerical Recipes in C*, Cambridge, England: Cambridge University Press.

Skolnik, M. I., 1980, *Introduction to Radar Systems,* New York: McGraw-Hill Book Company.

Skolnik, M. I., 1990, *Radar Handbook,* New York: McGraw-Hill Book Company.

Suresh Babu, B. N., and J. A. Torres, 1995, "Initial Results of Space-Time Processing Analysis of Multichannel Airborne Radar Measurements (MCARM) Data Files," *Third Annual Workshop on Adaptive Sensor Array Processing,* Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, MA.

Ward, J., 1994, *Space-Time Adaptive Processing for Airborne Radar,* Technical Report 1015, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, MA.

Appendix A

Filter Coefficients and Taper Weights

In this appendix, we describe the FIR lowpass filter coefficients, the pulse compression coefficients, and the weight tapers used in the application of the preprocessing and Doppler processing described in Sections 2 and 3 of this report for the MCARM data collection system. The following table contains the first 18 real-valued coefficients of the 36-tap FIR lowpass filter used to implement the I/Q conversion process. The final 18 coefficients obey the property: $h_a(35-n) = h_a(n)$, for $n = 0,...,17$.

Table A-1. FIR Lowpass Filter Coefficients

Tap Index	Coefficient
0	-1.6102325379139743e-03
1	-2.8646698676534879e-03
2	-3.0617070475842689e-03
3	-2.9357617638042620e-04
4	6.1262081463881951e-03
5	1.4294106583588592e-02
6	1.9275452700646348e-02
7	1.4737539657310698e-02
8	-3.3578280531961473e-03
9	-3.2523070782627228e-02
10	-6.1449167561121960e-02
11	-7.1877204855503404e-02
12	-4.5053503730479859e-02
13	2.9350592632137750e-02
14	1.4589099239674899e-01
15	2.8167135508217145e-01
16	4.0220175995940011e-01
17	4.7313563326838914e-01

An expression for the FM signal that forms the matched filter weights used to implement pulse compression is given by:

$$s_p(k) = \exp\left\{-j\pi B_s\left[\frac{1}{\tau_u}\left(\frac{k}{f_s}\right)^2 - \left(\frac{k}{f_s}\right)\right]\right\}\text{rect}\left(\frac{k}{f_s\tau_u}\right).$$

In the above expression, B_s represents the bandwidth of the transmitted signal and is equal to 1 MHz for the MCARM antenna. The symbol τ_u corresponds to the uncompressed pulsewidth of the signal and has a value of 50.4 microseconds. Finally, f_s represents the sampling rate after decimation and has a value of 1.25 MHz. These weights are computed off-line and combined with the taper weights to form the coefficients of the pulse compression filter. For this example, a Blackman taper is used. The coefficients of this taper are

$$w(k) = 0.42 - 0.5\cos\left(2\pi k/(K_p - 1)\right) + 0.08\cos\left(4\pi k/(K_p - 1)\right),$$

for $k = 0,1,\ldots,K_p - 1$, where K_p corresponds to the length of the pulse compression filter and equals $\lceil f_s\tau_u \rceil$.

A Blackman taper is also used to implement the Doppler processing. In this example, the taper has the form

$$d(p) = 0.42 - 0.5\cos(2\pi k/(P-1)) + 0.08\cos(4\pi k/(P-1)),$$

for $p = 0,1,\ldots,P-1$, where P corresponds to the number of pulses used in the Doppler processing block.

Appendix B

Single-Processor QR-Decomposition Benchmark

The QR-decomposition is a critical component of space-time adaptive processing (STAP). The number and size of the QR-decompositions required by a particular STAP algorithm largely determines the algorithm's computational complexity. When designing a parallel implementation for a real-time STAP application, a key point is whether the application's latency requirement forces a parallel implementation of the QR-decomposition, or whether the often multiple QR-decompositions can be accommodated across single nodes. A parallel implementation of the QR-decomposition implies an increase in the complexity of the parallel software and usually results in significantly reduced processing efficiency.

This single-processor QR-decomposition benchmark provides useful information that bears on this parallelization issue for matrices of practical sizes found in current and near-term STAP applications. The subsequent discussion provides a brief description of the QR-decomposition along with simple guidelines for implementing the algorithm in software. This discussion is followed by an outline of the reporting guidelines for the single-processor QR-decomposition benchmark.

Let $\vec{Y}$ correspond to an $m \times n$ matrix (m rows and n columns) of *complex* values, where $m > n$ and $\vec{Y}$ has full column rank. The QR-decomposition of this matrix is:

$$\vec{Y} = \vec{Q}\,\vec{R},$$

where $\vec{R}$ is an $m \times n$ upper triangular matrix and $\vec{Q}$ is an $m \times m$ unitary matrix. The matrix $\vec{R}$ can be written as $\left[\vec{R}_1^{\mathrm{T}} \quad \vec{0}\right]^{\mathrm{T}}$, where $\vec{R}_1$ is a $n \times n$ full rank upper triangular matrix. The QR-decomposition can be computed using a number of techniques including those based upon Householder and Givens transformations, along with the modified Gram-Schmidt approach[1]. These algorithms tradeoff computational complexity, storage requirements, and

1 Golub, G. H. and C. F. Van Loan, 1989, *Matrix Computations, 2nd Edition,* Baltimore, MD: The Johns Hopkins University Press, pp. 211-219.

other factors that impact the implementation and performance of the algorithm on high performance computers. For this benchmark, the user is allowed to select a particular algorithm for implementing the QR-decomposition. However, the algorithm selected must be suitable for decomposing complex matrices, and it must explicitly compute and store the upper triangular matrix $\vec{R}_1$, but not necessarily store the unitary matrix $\vec{Q}$.

As part of the RT_STAP sequential software package, we provide 12 single-precision complex data matrices, $\vec{Y}$, that can be used as inputs for evaluating the computational complexity of a given implementation of the QR-decomposition. The sizes of these data matrices and the names of the files in which they are stored are given in Table B-1 in two groups of the form $2n \times n$ and $4n \times n$. Note that the condition number[2] of $\vec{Y}^{H}\vec{Y}$ increases with n to provide an increasingly stressful test of the accuracy of the implementation. Standard single-precision floating-point precision should be sufficient in all cases. For these data files the matrix row index is chosen as the slowest running parameter. That is, for the 64×16 matrix the first 16 values correspond to the first row of the matrix, the second 16 correspond to the second row and so on for 64 rows.

The single-processor QR-decomposition benchmark is concerned with the processing time of software intended to run on a single processing node of a high performance computer. Usually this will correspond to a single microprocessor; however, some architectures contain processing nodes with multiple processors sharing memory. Such configurations are allowed, but the results should be reported when the code runs both on a single microprocessor and the full node. In all cases, the matrix should be initially stored in shared memory accessible to the network interface. It can be stored in either column or row major order.

To determine the processing time for each QR benchmark case, a time stamp t_s is calculated just before the QR-decomposition routine is called. A second time stamp t_c is calculated right after the corresponding final result is returned from this same function. The processing time for this problem instance is then $t_c - t_s$. Wall clock time should be used.

---

2 Golub, G. H. and C. F. Van Loan, 1989, *Matrix Computations, 2nd Edition,* Baltimore, MD: The Johns Hopkins University Press, pp. 79-81.

Table B-1. QR Benchmark Data Matrices and Weight Vectors

Size of Data Matrix $\vec{Y}$	Data Matrix File Name	Condition (dB) of Matrix $\vec{Y}^{H}\vec{Y}$	Weight Vector File Name
32 x 16	Y32x16.dat	75	W32x16.dat
64 x 32	Y64x32.dat	80	W64x32.dat
96 x 48	Y96x48.dat	85	W96x48.dat
128 x 64	Y128x64.dat	90	W128x64.dat
160 x 80	Y160x80.dat	95	W160x80.dat
192 x 96	Y192x96.dat	100	W192x96.dat
64 x 16	Y64x16.dat	75	W64x16.dat
128 x 32	Y128x32.dat	80	W128x32.dat
192 x 48	Y192x48.dat	85	W192x48.dat
256 x 64	Y256x64.dat	90	W256x64.dat
320 x 80	Y320x80.dat	95	W320x80.dat
384 x 96	Y384x96.dat	100	W384x96.dat

Each benchmark case consists of running 1000 trials. The average times should be reported for the 12 matrices provided as part of the RT_STAP benchmark software.

To verify that the QR-decomposition was computed correctly the software should solve the system of linear equations $\vec{R}_1^{H}\vec{R}_1\vec{w} = \vec{u}$ for the weight vector $\vec{w}$, where $\vec{u}$ is a $n \times 1$ vector whose entries are all one. The software should compare the elements of the weight vector computed by the QR-decomposition process with those generated by the LINPACK

software[3]. If we let $w(i)$ represent an element of the double-precision weight vector computed using the double-precision version of the LINPACK QR-decomposition, and let $\hat{w}(i)$ represent the same element computed using the software being benchmarked, then the following relative error criterion must be met to validate the software:

$$\max_i 10\log_{10}\left(\frac{|w(i)-\hat{w}(i)|^2}{|w(i)|^2}\right) < -10.$$

Table B-1 provides the names of the files containing the double-precision weight vectors computed using the double-precision version of the LINPACK software. All computations performed to validate the results must be done using double-precision values.

Table B-2 provides a template for reporting the results. The operation rate in millions of floating-point operations per second (Mflop/s) is computed by dividing the number of floating-point operations required to implement the QR-decomposition by the average time value. The number of floating-point operations for the QR-decomposition of an $m \times n$ complex matrix should be taken uniformly as $8mn^2$.

---

3 Dongarra, J. J., C. B. Moler, J. R. Bunch, G. W. Stewart, 1979, *LINPACK User's Guide* Philadelphia, PA: SIAM, pp. 9.1-9.25.

Table B-2. QR Benchmark Results Template

QR-Decomposition Algorithm:		
Microprocessor Name:		
Peak Operation Rate (Mflop/s):		
Matrix Size	**Average Processing Time (msec)**	**Operation Rate (Mflop/s)**
32 x 16		
64 x 32		
96 x 48		
128 x 64		
160 x 80		
192 x 96		
64 x 16		
128 x 32		
192 x 48		
256 x 64		
320 x 80		
384 x 96		

•U.S. GOVERNMENT PRINTING OFFICE: 1998-610-130-61134

MISSION OF ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.